PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46 — A1

(11) International Publication Number: WO 97/22927

(43) International Publication Date: 26 June 1997 (26.06.97)

(21) International Application Number: PCT/SE96/01706

(22) International Filing Date: 19 December 1996 (19.12.96)

(30) Priority Data:
08/574,977 19 December 1995 (19.12.95) US

(71) Applicant: TELEFONAKTIEBOLAGET LM ERICSSON (publ) [SE/SE]; S-126 25 Stockholm (SE).

(72) Inventor: RONSTRÖM, Mikael; Hägerstensvägen 119, 1tr., S-126 48 Hägersten (SE).

(74) Agents: BOHLIN, Björn et al.; Telefonaktiebolaget LM Ericsson, Patent and Trademark Dept., S-126 25 Stockholm (SE).

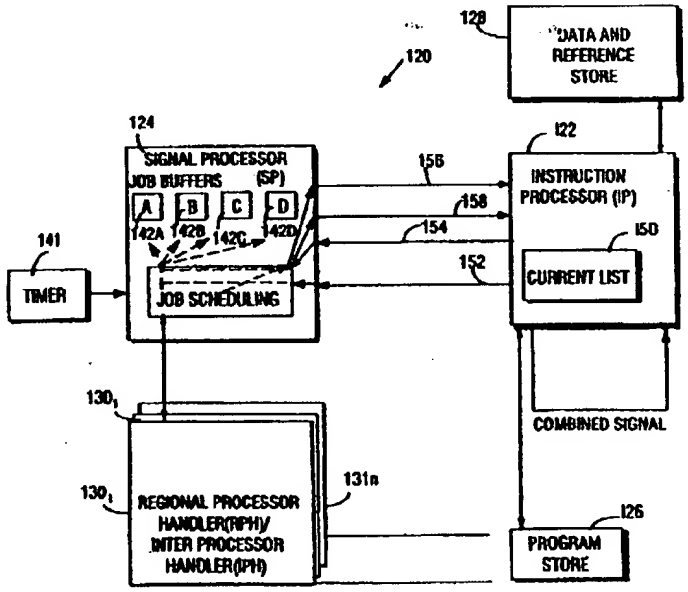
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: JOB SCHEDULING FOR INSTRUCTION PROCESSOR

(57) Abstract

In processing systems (120, 220), a signal processor (124, 224) schedules jobs to be executed by one or more instruction processor(s) (122, 222) and transmits a job-associated signal to the instruction processor when the instruction processor is to execute a job. A current list (150) is maintained in memory by the instruction processor (122). When a current job executed by the instruction processor causes the instruction processor to generate a buffered signal associated with a new job to be executed, the instruction processor selectively causes the buffered signal associated with the new job to be stored in the current list. Selective storage of the buffered signal in the current list is in accordance with a priority level of the current job. The new job is immediately executed upon termination of the current job if an instruction in the current job which generated the buffered signal is in a predefined order within the current job. When the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor selectively: (1) sends an EXIT signal to the signal processor [if the job associated with the buffered signal has a predefined priority level or if the current list is empty]; (2) sends all remaining jobs in the current list to the signal processor [if the signal processor has issued an interrupt to the instruction processor]; or (3) fetches and executes a further job from the current list.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgyzstan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

JOB SCHEDULING FOR INSTRUCTION PROCESSOR

BACKGROUND

1. Field of Invention

This invention pertains to the scheduling of jobs for execution by an instruction processor included in a central processing system.

2. Related Art and Other Considerations

Some processing systems, such as the Ericsson APZ 212 20, use a signal processor to schedule jobs for execution by an instruction processor, thereby enabling very rapid context switching between jobs executed by the instruction processor. The context switching time is very fast in such systems compared to most other processors.

In such systems, each job has a signal associated therewith. The signal contains information which advises the instruction processor as to which block of code (e.g., in a program store) should be executed by

the instruction processor in connection with the job, and data to be utilized in such execution. When a job has been executed by the instruction processor, a new signal (associated with a new job) is obtained by the signal processor and forwarded to the instruction processor. The new signal is obtained by the signal processor from a selected one of a plurality of job buffers of the signal processor, the new signal being obtained in priority order.

The new job obtained from the signal processor may be totally unrelated to the old job, and accordingly may use entirely different data. Moreover, other jobs are arriving from external sources (e.g., other processing systems) and, when of a higher priority, such jobs interrupt execution of the old job before the old job exits.

Thus, in processing systems as described above, the code and data changes context very often. This means that the context is lost so often that nothing is gained by using fast memories (e.g., cache memories) for saving the contexts. In some such systems, the only context save that is used is to store data in registers, which can only be done during execution of one job.

Considering such a system in more detail, Fig. 1 shows portions of a central processing unit 20, specifically a instruction processor unit (IP) 22; a signal processing (SP) unit 24; program store (PS) 26; data and reference store (DRS) 28; a plurality of regional processor bus handlers (RPHs) $30_{1,\ldots n}$; an

"other" processor bus handlers (IPH) 31, and, a maintenance unit (MAU) 32. Each of instruction processor 22, signal processor 24, IPHB 31, and RPHs 30 are independent processors. Instruction processor 22 executes jobs, each job corresponding to a block of instructions stored in program store (PS) 26. Signal processor 24 serves as a scheduler of jobs for instruction processor 22. In connection with such scheduling, for each job signal processor 24 receives a "signal", e.g., from the outside world or from instruction processor 22. A signal is an instruction telling where to execute in a specific part of a block of instructions, the signal including data to be utilized in execution of the block. Signal processor 24 analyses and prepares incoming signals, and assigns priority to these signals before they are sent to the instruction processor 22. A reference portion of data and reference store 28 contains information describing the signals, the blocks, and the variables used in the system.

In some configurations, central processing system 20 of Fig. 1 includes two instruction processors 22, two signal processors 24, one or other processor bus handlers (IPHs) 31, and a further plurality of regional processor bus handlers 30, all connected in a mirror image of Fig. 1 via MAU 32 and buses 34 and 36. In such configuration, each instruction processor 22 is provided with its own program store 26 and its own data and reference store 28.

Regional processor bus handlers (RPHs) $30_{1,...n}$ are connected by corresponding regional processor busses

$38_{1,...n}$ to unillustrated regional processors. Similarly, one or more other processor bus handlers 31 can also be provided for connection to a suitable bus 39. Both regional processor bus handlers (RPHs) $30_{1,...n}$ and other processor bus handlers, as well as signal processor 24, serve to decrease the load on instruction processor 22, as instruction processor 22 is involved in executing application software stored in program store 26.

Fig. 2 shows, in more detail, instruction processor 22, signal processor 24, other processor bus handler 31, and regional processor bus handlers (RPHs) $30_{1,...n}$ of central processing system 20, and interactions therebetween. In particular, Fig. 2 shows signal processor 24 as comprising a job scheduler 40 and a plurality of job buffers 42A - 42D, also known as buffers A - D, respectively.

In central processing system 20, when instruction processor 22 finishes a job (as indicated by sending an EXIT signal to signal processor 24), signal processor 24 retrieves a next job to be executed from the one of buffers 42A - 42D which has the highest priority and yet is non-empty. Further, signal processor 24 sends the signal associated with the job next to be executed to instruction processor 22. Receipt of such signal prompts instruction processor 22 to start its execution at the new block of code specified by the signal.

On occasion, the instructions executed by instruction processor 22 prompt instruction processor 22 itself to generate a new signal. Such instruction

processor-generated signals are of various types: combined signals, RP signals, other instruction processor signals, and buffered signals. A combined signal is much like a subroutine call, and results in instruction processor 22 immediately executing the combined signal and then returning to execute the job from which the combined signal was generated.

The RP signals and other instruction processor signals generated by instruction processor 22 are signals associated with jobs to be executed either by a regional processor (in the case of RP signals) or another instruction processor (in the case of an IP signal). An RP signal or other IP signal is received by signal processor 24 and is directed to an appropriate one of the regional processor bus handlers 30 or (in the case of an other IP signal) to an other processor bus handler 31 (in the case of an RP signal).

An IP-generated signal, other than a combined signal, which is generated by instruction processor 22 for execution by instruction processor 22, is referred to as a "buffered signal". In accordance with the prior art, the buffered signal is sent to signal processor 24 (as represented by line 54 in Fig. 2). Execution by instruction processor 22 continues in the block instruction processor 22 is currently executing. Upon receipt of such a buffered signal emanating from instruction processor 22, signal processor 24 takes one of two potential actions depending upon priority level of the received buffered signal. In particular, if the priority level of the received buffered signal is greater

than the priority level of the job currently being executed by instruction processor 22 (i.e., the job which generated the buffered signal), then signal processor 24 sends an interrupt to instruction processor 22 to interrupt the current job (as represented by line 56 in Fig. 2). Otherwise, signal processor 24 puts the buffered signal as the last job in the one of the job buffers 42A - 42D having the same priority as the buffered signal. In such manner, the buffered signal is executed as any other signal of the same priority as the buffered signal (as represented by line 58 in Fig. 2).

Thus, in the prior art as described above, when an EXIT instruction is executed by instruction processor 22 at the end of a job, the instruction processor 22 always looks to signal processor 24 for scheduling of the next job. In such scheduling, signal processor 24 fetches the next job from job buffers 42A - 42D in priority order. However, given this prior art scheduling regime, the job so fetched can be totally unrelated to the previous job. Likely such an unrelated job would use mostly different data. Also, if jobs arrive with a higher priority, they might interrupt execution of a job before an EXIT instruction is reached.

Central processing system 20 has numerous implementations including, for example, as a control system for a telephone switching system. As mentioned above, one example of such implementation is the APZ 212 20 control system for the Ericsson AXE 10 switch, as described by Egeland, Terje, "APZ 21220 -- The New High-

end Processor for AXE 10", Ericsson Review, No. 1, 1995, pp. 5 - 12, which is incorporated herein by reference.

What is needed is for such a central processing system is a way to not change context so often, and thereby effectively use very fast memory such as cache memory. Other desired improvements would be relieving the signal processor from some of its work.

SUMMARY

Scheduling of signals for execution by an instruction processor is primarily performed by a signal processor. In accordance with the present invention, the signal processor has a unique scheduling technique and the instruction processor performs some of its own scheduling (using a "current list" memory) without disturbing the signal processor.

The signal processor allocates signals to be executed into one of four buffers according to an assigned priority of the signal, e.g., into scheduling buffer A, scheduling buffer B, scheduling buffer C, or scheduling buffer D. In general, when an instruction processor needs a new signal for execution from the signal processor (as occurs upon receiving an EXIT signal from the instruction processor), the signal is fetched from buffers A - D in accordance with age (oldest) and priority level (i.e., level A, B, C, or D). (As used herein, a designation of "highest" priority level does not take into consideration a trace level).

If the priority level of the fetched job is either of the "C" priority level or "D" priority level, the signal processor determines whether any jobs of similar priority level are currently interrupted and, if so, resumes execution of the interrupted job while sending the fetched job back to the appropriate one of the its buffers. Moreover, should the instruction processor be executing a level D signal at the time the signal processor receives a signal of higher priority (e.g, from the instruction processor, from a regional processor, or from an "other" instruction processor), execution of the level D signal by the instruction processor is immediately interrupted.

Execution of a signal by the instruction processor (IP) may cause generation of a new signal. Such instruction processor-generated signals are typically generated just before the EXIT instruction of a signal. Such instruction processor-generated signals can be combined signals or buffered signals (both of which are to be executed by the instruction processor) or a signal destined for another processor (such as a regional processor or an "other" instruction processor). A combined signal is much like a subroutine call, and results in the instruction processor immediately executing the combined signal and then returning to execute the signal in which the combined signal was generated.

When a buffered signal is generated by the instruction processor, the instruction processor puts the buffered signal in a special register or queue known as

the "current list". If the buffered signal happens to be generated immediately before exit of the currently executing job, is of any but the lowest priority level, and if no interrupt is set, the job associated with the buffered signal is executed upon exiting from the job in which the buffered signal is generated.

Subsequently, when the instruction processor exits from a job executed as a result of generation of a buffered signal, the instruction processor can perform any one of several alternative steps depending upon the priority level of the buffered signal and depending upon whether any interrupts have been set. A first such alternative includes obtaining the first (next) job from the current list and executing the same. A second such alternative step includes returning scheduling control to the signal processor, as occurs when the buffered signal is of a lowest priority level. A third such alternative step includes transferring the entire contents of the current list to the signal processor, as occurs when an interrupt has occurred. In connection with the first alternative, consecutive jobs on the current list can be consecutively executed as jobs on the current list are exited.

Whether or not a job associated with a buffered signal is executed immediately after the job which generates the buffered signal depends on the whether the buffered signal is generated immediately before a predetermined type of instruction (e.g, an EXIT instruction). Thus, an instruction causing generation of a buffered signal must be in a predefined order within a

job to have the buffered signal executed immediately upon termination of the job. Otherwise, the job associate with the buffered signal is subject to remaining on the current list and being executed subsequently to jobs associated with other buffered signals, or even being transferred to the signal processor.

Whenever execution of a job by the instruction processor is interrupted, the entire contents of the current list is transferred to the signal processor. The signal processor then puts the signals from the current list into appropriate ones of its buffers in accordance with priority levels of the transferred signals.

The signal processor also has the capability of sending signals to the current list. When the signal processor transmits a signal to the instruction processor for execution, the signal processor searches its buffer(s) and transfers to the current list another signal in its buffer(s) that has a same thread identification as the signal being transferred to the instruction processor. A similar posting of a signal on the current list occurs when
the signal processor determines that the thread ID of the externally-generated signal (e.g., received via either a regional processor bus handler or a processor bus handler) is the CurrentThreadID being processed by the instruction processor but did not result in an interrupt or setting of an interrupt flag.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features, and advantages of the invention will be apparent from the following more particular description of preferred embodiments as illustrated in the accompanying drawings in which reference characters refer to the same parts throughout the various views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

Fig. 1 is a schematic diagram of at least a portion of a central processing unit.

Fig. 2 is a schematic diagram showing various constituent elements provided for a central processing system in accordance with the prior art.

Fig. 3 is a schematic diagram showing various constituent elements provided for a central processing system in accordance with an embodiment of the present invention.

Fig. 4 is a schematic diagram showing one illustration of a configuration of an instruction processor provided for the central processing system of Fig. 3.

Fig. 5 is a schematic diagram showing various constituent elements provided for a multi-instruction processor-based central processing system in accordance with another embodiment of the present invention.

Fig. 6 is a schematic diagram depicting events encountered and states experienced by a signal processor of the central processing system of Fig. 3.

Fig. 7 is a schematic diagram depicting events encountered and states experienced by an instruction processor of the central processing system of Fig. 3.

Fig. 8(1) is a flowchart showing steps executed by a signal processor upon encountering an EXIT signal from an instruction processor.

Fig. 8(2) is a flowchart showing steps executed by a signal processor upon encountering a (non-exit) signal from an instruction processor.

Fig. 8(3) is a flowchart showing steps executed by a signal processor upon encountering time out event.

Fig. 8(4) is a flowchart showing steps executed by a signal processor upon encountering a signal from a regional processor (RP) or other instruction processor.

Fig. 9(1) through Fig. 9(3) are schematic views showing steps executed by an instruction processor for performing actions A(SP)1 through A(SP)3, respectively.

Fig. 9(4) is a flowchart showing steps performed by an instruction processor for performing action A(SP)4.

Fig. 9(5) through Fig. 9(10) are schematic views showing steps executed by an instruction processor for performing actions A(SP)5 through A(SP)10, respectively.

Fig. 9(11) is a flowchart showing steps performed by an instruction processor for performing action A(SP)11

Fig. 10 is a schematic view of a format of a signal utilized by the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS

Central processing system 120 of Fig. 3 includes instruction processor unit (IP) 122; a signal processing (SP) unit 124; program store (PS) 126; data and reference store (DRS) 128; a plurality of regional processor bus handlers (RPHs) $130_{1,\ldots n}$; and, "other" processor bus handler(s) (IPB) 131. Unless stated explicitly or implicit otherwise, similarly named elements of system 120 are the same as the afore-described elements of system 20 of Fig. 2. Among the differences of system 120 and system 20 is a current list memory 150 which is stored in a set of registers of instruction processor 122 for access by instruction processor 122. In addition, signal processor 124 has a timer 141.

Central processing system 120 of Fig. 3 is realized in the illustrated embodiment utilizing a Sun Ultra 2 workstation, which provides two processors working with shared memory. The persons skilled in the

art understands how to send signals between two processors through shared memory techniques.

Instruction processor 122 and signal processor 124 operate in a different manner than do corresponding processors of the prior art. In particular, as seen hereinafter in more detail, instruction processor 122 selectively stores buffered signals which it generates on its current list 150, thereby bypassing signal processor 124 and providing better context preservation. Signal processor 124 allows one or more buffered signals in current list 150 to be executed so long as a time limit is not exceeded. Further, when signal processor 124 has a job of a higher priority level, signal processor 124 always interrupts "D" priority level jobs in the middle of job execution by instruction processor 122. Signal processor 124 further allows "A" priority level jobs to interrupt other jobs upon signal sending.

Fig. 4 shows one configuration of instruction processor 122. In the configuration of Fig. 4, instruction processor 122 has central processing unit (CPU) 160; a register memory 162; fast memory 164; memory access/interface 166; a plurality of cache memories 168A - 168C; a plurality of memory cards 170A - 170G; and, a memory bus 172 for connecting memory access/interface 166 to cards 170. In the configuration shown in Fig. 4, cache memory 168A has a 128-byte access; cache memory 168B has a 16-byte access; and, cache memory 168C has a 4-byte access. Cards 170A - 170F are DRAM main memory cards. Card 170G is a direct memory access (DMA) card connected by IOB bus to peripheral devices such as disk

drive controllers, making it possible to uses other memory devices such as disks.

In the instruction processor configuration of Fig. 4, fast memory 164 is utilized to access variables that are very frequently used, such as data structures for locking and for keeping track of disk buffers. An allocation of variables to specific memory types can be accomplished through the compiler or by the designer. Cache memories 168A - 168C have varying line sizes to offer flexibility. Variables which should be cached are also specifiable as a block parameter or variable type.

Fig. 10 shows the format of a signal utilized by one embodiment of the present invention. As shown in Fig. 10, the signal includes a first field "ThreadID" indicative of the thread to which the signal belongs; a second field "JobLevel" indicative of the priority level of the signal; a third field "FORMAT" indicated of the number of data items in the last field of the signal; a fourth field "SignalNumber" indicative of a sequence number of the signal; a sixth field "RecBlock" which is the receiving block number; and seventh field "SendBlock" which is the sending block number; an eighth field "Forlopp-Id" which is an identification utilized for reliability purposes; a ninth field "RegPRO" which contains a first data value; and, a tenth field "DataList" which contains one or more (as many as twenty four) other data values.

A thread includes the notion of a set of jobs which are necessary to execute in response to a message

received from an outside system (e.g., from a regional processor or an "other" instruction processor). In a thread, several jobs can be executing simultaneously and communication with outside systems can be occurring as part of the thread.

OPERATION

Events noted and actions taken by signal processor 124 are shown in Fig. 6. Events noted and actions taken by instruction processor 122 are shown in Fig. 7. Events noted by signal processor 124 are denoted with identifiers of the form "E(SP)x"; events noted by instruction processor 122 are denoted with identifiers of the form "E(IP)x". Actions taken by signal processor 124 are denoted with identifiers of the form "A(SP)x"; actions taken by instruction processor 122 are denoted with identifiers of the form "A(IP)x". In all such identifiers, "x" refers to the event or action number.

As shown in Fig. 6, signal processor 124 has three states: an IDLE state; a WORKING state; and a QUEUED state. Changes of state are shown by boldface lines in Fig. 6. Fig. 6 also shows that signal processor 124 encounters four "SP" events, in particular events E(SP)1 through E(SP)4. Signal processor 124 can be in any one of its three states when it encounters an SP event. For this reason, each SP event in Fig. 6 is shown incident upon each state (by broken lines).

For each SP event of Fig. 6, signal processor 124 responds with a corresponding action. As used herein, an "action" can include one or more steps, which

steps can be executed either alternatively or successively. Actions performed by signal processor 124, and the steps constituting each action, are illustrated in Fig. 8(1) through 8(4), respectively. At times, the steps executed pursuant to such actions depend upon the "state" of signal processor 124.

As shown in Fig. 7, instruction processor 122 has three states: an IDLE state; a WORKING state; and a COMBINED/WORKING state. Similarly named states of instruction processor 122 and signal processor 124 are not to be confused, as each state is independent. As in Fig. 6, changes of state are shown by boldface lines in Fig. 7. Fig. 7 also shows that signal processor 124 encounters eleven "IP" events, in particular events E(IP)1 through E(IP)11. Some of these IP events, e.g., IP events E(IP)1, E(IP)3, E(IP)9, occur in connection with actions performed by signal processor 124.

Signal processor 124 and instruction processor 122 are correlated so that certain IP events occur only when instruction processor 124 is in one or more specified ones of its states. In this regard, actions performed by instruction processor 122 are shown within circles depicting the states in which the actions may occur. States in which the IP events are encounterable are indicated by broken line connections in Fig. 7. Steps involved with IP actions E(IP)1 through E(IP)11 are shown in more detail in Figs. 9(1) - 9(11), respectively. As understood with respect to Figs. 9(1) - 9(11) as described below, various actions performed by instruction

processor 122 create the SP events E(SP)1 - E(SP)2 shown in Fig. 6.

A. SIGNAL PROCESSOR OPERATION

Signal processor 124 operates in conjunction with its timer 141 (see Fig. 3). When loaded with a particular time value (e.g, value "X" or "Y"), timer 141 notifies signal processor 124 when a time interval corresponding to the time value has expired. Expiration of the time value, e.g., notification by timer 141, causes a time out event [event E(SP)3], described in more detail with reference to Fig. 8(3). Time out events enable signal processor 124 to know whether an action of instruction processor 122 is taking too much time. In this regard, upon initiation of certain operations (e.g., transmitting a new signal from a buffer 142 of signal processor 124 for execution by instruction processor 122) timer 141 is set for a value "X" (e.g., 1 ms). If a time out occurs without timer 141 having been reset to value "X", signal processor 124 sets the flag SP_Interrupt to note that instruction processor 122 has been executing jobs of the same thread ID for one time interval. Timer 141 is then loaded with a second time value "Y" (e.g., 3 ms). Action taken, should a second time out occur after expiration of time value "Y", depends on the priority level of the executing job. If the executing job is of "C" or "D" priority level, the executing job is interrupted. If the executing job is of "A" or "B" priority level, an error condition is noted an a KILL signal is issued.

Actions performed by signal processor 124 upon receipt of SP events are described below:

(1) RECEIPT OF NON-EXIT SIGNAL BY SP

In connection with its execution of jobs, instruction processor 122 may generate a non-exit signal and assign a priority level for the generated signal. In the case that such a signal is communicated to signal processor 124, signal processor 124 sees event E(SP)2 as shown in Fig. 6. Upon receipt of event E(SP)2, signal processor 124 performs action A(SP)2, the steps of which are illustrated in Fig. 8(2).

At respective steps 8(2)-1 and 8(2)-2, signal processor 124 discerns whether the signal received from instruction processor 122 is destined for one of the regional processors or an "other" instruction processor. If the signal received from instruction processor 122 is destined for one of the regional processors, at step 8(2)-3 the signal is sent to an appropriate regional processor bus handler 130. If the signal received from instruction processor 122 is destined for one of an "other" instruction processor, at step 8(2)-4 the signal is sent to "other" instruction processor bus handler 131.

If the non-exit signal obtained from instruction processor 122 is not externally destined, the priority level of the signal currently being executed by instruction processor 122 is checked at step 8(2)-5. If the priority level of the currently executing signal is other than "D" level, at step 8(2)-6 the received signal is put last in an appropriate job buffer 42A - 42D in

accordance with its priority. Otherwise, if the priority level of the currently executing signal is "D" level, then step 8(2)-7 is executed. At step 8(2)-7, switch Time-out is set to value "X"; flag Job_interrupted_level[D] is set active (to indicate that a job of priority level D is being interrupted); and an execution of the current ("D" level) job is interrupted. Instruction processor 122 sees the job interrupt as event E(IP)8, and responds with action A(IP)8 as described in Fig. 9(8). Upon completion of either of steps 8(2)-3, 8(2)-4, 8(2)-6, or 8(2)-7, action A(SP)2 terminates as indicated by symbol 8(2)-8.

(2) RECEIPT OF EXIT SIGNAL BY SP

Signal processor 124 receiving an EXIT signal [event E(SP)1] from instruction processor 122 triggers Action A(SP)1, steps of which are shown in Fig. 8(1). An EXIT signal may be received from instruction processor 122, and accordingly action A(SP)1 triggered, in the instances of instruction processor 122 completing execution of a job that does not result from generation of a buffered signal. Alternatively, an EXIT signal may be received when instruction processor 122 completes a job or sequence of jobs which did result from a buffered signal [see steps 9(4)-6, 9(4)-9, and 9(4)-4 in Fig. 9(4); steps 9(7)-3 in Fig. 9(7); and step 9(3)-3 in Fig. 9(3)].

Upon receipt of an EXIT signal from instruction processor 122 [Event E(SP)1], signal processor 124 fetches the first job from the non-empty buffer 142A - 142D with the highest priority [step 8(1)-1]. At step

8(1)-2, signal processor 122 clears various flags (Error_timeout and SP_Interrupt) and sets switch Time-out to a value "X". At step 8(1)-3 and step 8(1)-4, the priority level of the job fetched at step 8(1)-1 is checked to determine whether the priority level is either "C" (step 8(1)-3) or "D" (step 8(1)-4).

If the fetched job has priority level "C", then at step 8(1)-5 a flag Job_interrupted_level[C] is consulted to determine if the flag is active, i.e., whether execution by instruction processor 122 of another job of priority level C has already been interrupted by signal processor. If the determination at step 9(1)-5 is affirmative, step 8(1)-6 is executed. In step 8(6)-1, the job fetched at step 8(1)-1 is put back into the buffer from whence it was fetched; flag Job_interrupted_level(C) is set to be not active (since the previously interrupted level "C" job will now be executed and therefore is uninterrupted); a flag Active_priority is set to "C" (to re-establish the priority level of the previously interrupted job); and the interrupted job is resumed. Resumption of interrupted job execution by signal processor 124 creates IP event B(IP)2, understood with reference to Fig. 9(2). Action A(SP)1 then terminates (as indicated by symbol 8(1)-9).

If the fetched job has priority level "C", but the flag Job_interrupted_level[C] is not active (i.e., if no job of priority level "C" is currently interrupted), step 8(1)-7 and step 8(1)-8 are executed. At step 8(1)-7, flag Active_priority is set to be the priority of the

job fetched at step 8(1)-1; a signal associated with the fetched job is sent as IP event E(IP)1 to instruction processor 122 (see Fig. 9(1)); and, CurrentThreadID is set to the thread ID of the fetched job. At step 8(1)-8, the thread IDs of all jobs in the job buffers of priority level "C" or higher are checked and, if a job's thread ID is the same as the CurrentThreadID (i.e, the thread ID of the fetched job), then a signal associated with each such job is sent to current list 150 in instruction processor 122. In other words, an IP event E(IP)11 is generated for each such job in the job buffer(s) (of level "C" or higher) which has a thread ID which is the same as CurrentThreadID. Action taken by instruction processor 122 in response to IP event E(IP)11 is described with reference to Fig. 9(11). Action A(SP)1 then terminates (as indicated by symbol 8(1)-9).

If the fetched job has priority level "D" (as determined at step 8(1)-4), then at step 8(1)-10 flags Job_interrupted_level[C] and Job_interrupted_level[C] are consulted to determine if execution by instruction processor 122 of jobs of either priority level "C" or "D" are currently interrupted by signal processor 124. If the check at step 8(1)-10 is affirmative (i.e., if either flag is active), the job fetched at step 8(1)-1 is put back in the buffer [step 8(1)-11] and a further check is preformed at step 8(1)-12. Specifically, at step 8(1)-12, a further discrimination is conducted to see if flag Job_interrupted_level[C] is active. If the result of step 8(1)-12 is affirmative, at step 8(1)-13 flag Job_interrupted_level[C] is set to be not active and flag Active_Priority is assigned the value "C". If the result

of step 8(1)-12 is negative, at step 8(1)-14 flag Job_interrupted_level[D] is set to be not active and flag Active_Priority is assigned the value "D". Upon completion of either steps 8(1)-13 and 8(1)-14, at step 8(1)-15 signal processor 124 directs instruction processor 122 to resume execution of the interrupted job of highest priority by generating IP event E(IP)2. Thereafter, action A(SP)1 terminates as indicated by symbol 8(1)-16.

If it were determined at step 8(1)-10 that neither flag Job_interrupted_level[C] or Job_interrupted_level[D] were active, then steps 8(1)-17 and 8(1)-18 are executed prior to termination of action A(SP)1. At step 8(1)-18, flag Active_priority is set to be the priority of the job fetched at step 8(1)-1; CurrentThreadID is set to the thread ID of the fetched job; and, a signal associated with the fetched job is sent as IP event E(IP)1 to instruction processor 122 (see Fig. 9(1)), in similar manner to step 8(1)-7. At step 8(1)-18, the thread IDs of all jobs in the job buffers (having priority level of "C" or higher) are checked and, if a job's thread ID is the same as the CurrentThreadID (i.e, the thread ID of the fetched job), then a signal associated with each such job is sent to current list 150 in instruction processor 122. In other words, a IP event E(IP)11 is generated for each such job in the job buffer which has a thread ID which is the same as CurrentThreadID. As indicated previously, action taken by instruction processor 122 in response to IP event E(IP)11 is described with reference to Fig. 9(11).

If the fetched job has a priority level other than "C" or "D", then steps 8(1)-19 and 8(1)-20 are executed prior to termination of action A(SP)1. Steps 8(1)-19 and 8(1)-20 are similar to steps 8(1)-17 and 8(1)-18, and essentially result in all jobs in the buffer which have a thread ID which is the same as the CurrentThreadID being sent to the instruction processor's current list 150. Termination of action A(SP)1 is indicated by symbol 8(1)-21.

(3) RECEIPT OF SIGNAL FROM RP OR OTHER IP BY SP

Just as signal processor 124 can send signals to regional processor bus handlers 30 and the "other" instruction processor bus handler 31, signal processor 124 can receive signals externally generated from those handlers, as indicated by Event E(SP)4 in Fig. 6. Upon receipt of such an externally-generated signal, Action A(SP)4 is preformed. The steps of action A(SP)4 are illustrated in Fig. 8(4).

In Action A(SP)4, if the externally-generated signal does not contain a thread ID (as determined at step 8(4)-1), signal processor 124 assigns a thread ID [step 8(4)-2]. For example, signal processor 124 can assign a thread ID by concatenating a processor ID and an internal counter that is incremented each time a new thread ID is assigned.

In steps 8(4)-3 and 8(4)-4, signal processor 124 examines the priority level of the externally-generated signal. If conditions of neither step 8(4)-3

nor step 8(4)-4 are satisfied, a thread ID condition of step 8(4)-5 is checked. If the thread ID condition of step 8(4)-5 is not satisfied, step 8(4)-6 is executed.

If, in step A(SP)4-3, the priority level of the externally-generated signal is greater than the priority level of the job currently being executing by instruction processor 122, and the job being executed by instruction processor 122 has the lowest ("D") level priority, signal processor 124 performs step 8(4)-7. In step 8(4)-7, signal processor 124 sets switch Time-out to value "X"; sets flag Job_interrupted_level[D] as active (to indicate that a job of priority level "D" is interrupted); and, sends an interrupt to instruction processor 122. Instruction processor 122 views the interrupt as IP event E(IP)8, a response to which is understood with respect to action A(IP)8 described in Fig. 9(8).

If, in step 8(4)-4, the priority level of the externally-generated signal is the highest level (e.g., "A" level priority) and is greater than the priority level of the job currently being executing by instruction processor 122, at step 8(4)-8 signal processor 124 sets a flag SP_Interrupt (which causes instruction processor 122 to execute the new signal upon termination of the job currently being executed). Further, at step 8(4)-7, signal processor 124 sets the value of switch Time-out to "Y".

Assuming that the externally-generated signal does not result in interruption or in setting flag SP_Interrupt, at step 8(SP)4-5 signal processor 124

nevertheless determines if the thread ID of the externally-generated signal (received via either a regional processor bus handler 30 or a processor bus handler 31) is the CurrentThreadID being processed by instruction processor 122 and if the priority level of the externally-generated signal is greater than D-level. If so, at step 8(4)-9 the externally-generated signal is sent to instruction processor 122 for entry onto current list 150. Upon receipt of such a signal [Event E(IP)9], instruction processor 122 puts the externally-generated signal as the last entry in current list 150 (see Fig. 9(9)).

If none of steps 8(4)-3 through 8(4)-5 are executed, then at step 8(4)-6 signal processor 124 puts the externally-generated signal in the appropriate one of job buffers 142A - 142D, according to the priority level of the externally-generated signal. Upon completion of any of steps 8(4)-7, 8(4)-8, 8(4)-9, or 8(4)-6, action A(SP)4 terminates as indicated by symbol 8(4)-10.

(4) TIME-OUT FOR SP

Timer 41 can have a time out when the value stored for switch Time-out is clocked down to zero by timer 41. When timer 141 has a time out, signal processor 124 sees the time out as event E(SP)3 (see Fig. 6) and performs action A(SP)3, and particularly the steps shown in Fig. 8(3). Upon reaching a time out, signal processor 124 first checks at step 8(3)-1 whether the flag SP_Interrupt has been set. If the flag has not been set, at step 8(3)-2 timer 141 (Time-out) is set to value "Y" and flag SP_Interrupt is set before action A(SP)3 is

terminated (as indicated by symbol 8(3)-3). If the flag has not been set, step 8(3)-4 is next executed.

At step 8(3)-4 the flag SP_Interrupt is cleared. Then, at step 8(3)-5 a determination is made whether the job currently being executed by instruction processor 122 has a priority level of "C" or "D". If the determination at step 8(3)-5 is affirmative, step 8(3)-6, step 8(3)-7, and step 8(3)-8 are consecutively executed prior to termination of action A(SP)3. At step 8(3)-6, signal processor 124 fetches a first (oldest) job from a non-empty one of buffers 142A - 142D having the highest priority. Then, at step 8(3)-7, the flag Job_interrupted_level[Active_priority] is set active. In other words, signal processor 124 notes that the time out will cause interruption of a currently executing job (having the priority level "Active_priority"), and accordingly that the flag Job_interrupted_level must be set for the priority level of the interrupted job. At step 8(3)-8, the job currently being executed by instruction processor 122 (and during whose execution the time out occurred) is interrupted. Such interruption is seen by instruction processor 122 as IP event E(IP)8, a response to which is understood in connection with action A(IP)8 as described in Fig. 9(8).

If the determination at step 8(3)-5 is negative, i.e, if the priority level of the job currently being executed by instruction processor 122 is higher than "C" level, at step 8(3)-10 a signal KILL_SIGNAL is sent to instruction processor 122. Instruction processor views the signal KILL_SIGNAL as IP event E(IP)3, and

develops a responsive action A(IP)3 as shown in Fig. 9(3). Upon conclusion of step 8(3)-8 and step 8(3)-10, action A(SP)3 is terminated as indicated by symbol 8(3)-3.

B. INSTRUCTION PROCESSOR OPERATION

As mentioned previously, in the course of executing its jobs in its various states as shown in Fig. 7, Instruction processor 122 can itself generate signals. Such IP-generated signals include signals destined for regional processors or other instruction processors, combined signals, and buffered signals. Events caused by IP-generated signals are shown in Fig. 7, and include event E(IP)11 corresponding to generation of a buffered signal; event E(IP)10 corresponding to generation of a combined signal or a HURRY signal; event E(IP)6 corresponding to generation of a signal for a regional processor or an "other" instruction processor. A HURRY signal is executed immediately by instruction processor 122 without consulting signal processor 124, and involves instruction processor 122 changing to a new block and commencing execution in the new block. Thus, unlike a combined signal, a HURRY signal implies an exit without a return to the calling block.

Actions performed by instruction processor 122 in response to IP events are as follows:

(1) IP RECEIVING SIGNAL FROM SP

When in its IDLE state, instruction processor 122 performs action A(IP)1 shown in Fig. 9(1). In

particular, at step 9(1)-1 instruction processor 122 changes its state from IDLE to WORKING (see Fig. 7). Then, at step 9(1)-2, instruction processor 122 changes to a new block in its associated program store 126 and starts execution in the new block.

(2) IP JOB EXECUTION INTERRUPTED

Upon receipt of an interrupt [Event E(IP)8], instruction processor 122 takes action A(IP)8 shown in Fig. 9(8). At step 9(8)-1, instruction processor 122 saves its context. That is, instruction processor saves (e.g., in register memory 162) the block which was interrupted, its program counter, and the contents of its registers upon interruption. Then, at step 9(8)-2, instruction processor 122 sends a signal to signal processor 124 for each job stored in current list 150, thereby effectively transferring the contents of current list 150 to signal processor 124. Thereafter, at step 9(8)-3 instruction processor 122 changes to the new priority level of the interrupting signal and executes the job associated with the interrupting signal.

(3) IP RESUMING INTERRUPTED JOB

When instruction processor 122 is to resume execution of an interrupted job, event E(IP)2 is encountered. Action A(IP)2 taken in response to event E(IP)2 is shown in Fig. 9(2). At step 9(2)-1, instruction processor 122 restores the context of the interrupted job (from the register memory 162 as understood from the foregoing). Then, at step 9(2)-2, instruction processor 122 changes to the priority level

of the restored job and restarts execution of the signal associated with the restored job.

(4) IP SENDING SIGNAL TO RP OR OTHER IP

When the code executed by instruction processor 122 requires generation of a signal destined for a regional processor or an "other" instruction processor, event E(IP)6 occurs. Actions taken by instruction processor 122 in response to IP event E(IP)6 are shown in Fig. 9(6). In action A(IP)6, instruction processor 122 merely sends such generated signal to signal processor 124 and continues execution in the currently executing block (e.g., the block from which either the regional processor or "other" instruction processor signal was generated).

(5) IP SENDING COMBINED OR HURRY SIGNAL

When the code executed by instruction processor 122 requires generation of a combined signal or HURRY signal [Event E(IP)10], instruction processor 122 immediately sets its state to COMBINED [see Fig. 7 and step 9(10)-1 of Fig. 9(10)]. Then, at step 9(10)-2 instruction processor 122 saves the calling block (i.e., the block of instructions in which the Combined or HURRY signal was generated) and puts the return address for the calling block on the top of its stack. At step 9(10)-3 instruction processor 122 changes to the new block required for execution by the combined or HURRY signal and begins execution at the beginning of that new block.

(6) IP EXITING COMBINED OR HURRY SIGNAL

When instruction processor 122 encounters an EXIT instruction of a combined or HURRY signal, event E(IP)5 is encountered. Upon occurrence of event E(IP)5, action A(IP)5 as depicted in Fig. 9(5) is taken. At step 9(5)-1, instruction processor 122 restores the calling block (i.e., the block in which the combined or HURRY signal was generated) and returns the address of the next instruction for execution in the calling block from the stack of instruction processor 122. At step 9(5)-2 instruction processor 122 checks whether a flag "last Combined_Signal_return" has been set so as to indicate that the stack is empty and, if so, sets the state of instruction processor 122 to the WORKING state. Then, at step 9(5)-3, instruction processor 122 resumes execution of the calling block at the instruction specified by the stack address.

(7) IP SENDING BUFFERED SIGNAL

When an instruction executed by instruction processor 122 calls for generation of a buffered signal [Event E(IP)11], instruction processor 122 responds with action A(IP)11 shown in Fig. 9(11). At step 9(11)-1 the priority level of the currently executing job is evaluated to determine if it is of the lowest ("D") level. If so, at step 9(11)-2, the buffered signal is sent to signal processor 124 as event E(SP)2 [see Fig. 6] and execution in the buffer signal-generating block continues (as indicated by step 9(11)-3) as action A(IP)11 terminates (as indicated by symbol 9(11)-4.

If the priority level of the currently executing job has a priority level other than the lowest priority level, it is determined at step 9(11)-5 whether the next instruction to be executed is an EXIT instruction and whether the flag SP_Interrupt has not been set. If both conditions of step 9(11)-5 are affirmative, then (as indicated by step 9(11)-6) the exit instruction is merged with and performed by instruction processor 122 as part of the signal sending instruction. Otherwise, if either of the conditions of step 9(11)-5 are negative, instruction processor 122 puts the job associated with the buffered signal as the last job on current list 150 (step 9(11)-7). Upon completion of either step 9(11)-6 or step 9(11)-7, action A(IP)11 terminates as indicated by symbol 9(11)-4.

(8) IP EXITING BUFFERED SIGNAL

As instruction processor 122 executes a job associated with a buffered signal, instruction processor 122 eventually encounters an EXIT instruction [Event E(IP)4]. Upon encountering an EXIT instruction of a buffered-signal associated job, instruction processor 122 performs action A(IP)4 depicted in Fig. 9(4). A "buffered signal" not only includes a signal generated by the instruction processor 122 in the manner aforedescribed, but also encompasses all signals scheduled by signal processor 124 for instruction processor 122.

In connection with action A(IP)4, instruction processor 122 conducts the checks indicated by steps 9(4)-1 through 9(4)-3 and, if all such checks are

negative, executes step 9(4)-4 before action A(IP)4 is terminated (as indicated by symbol 9(4)-5).

At step 9(4)-1, instruction processor 122 determines if the job being executed is of the lowest priority level ("D" level). If so, at step 9(4)-6 instruction processor 122 sets its state to IDLE, and then sends an EXIT signal to signal processor 124 and is handled by signal processor 124 (see Event E(SP)1 in Fig. 6). Any remaining signals stored on current list 150 are not executed at this point.

At step 9(4)-2, instruction processor 122 checks whether a flag SP_Interrupt has been set. SP_Interrupt can be set at step 8(4)-7 of Fig. 8(4) when an externally-generated signal has a highest level priority and the currently executing job does not, or when a time-out has occurred [Action A(SP)3, step 8(3)-2]. If the determination at step 9(4)-2 is affirmative, steps 9(4)-7 through 9(4)-9 are executed prior to termination of action A(IP)4.

At step 9(4)-7, instruction processor 122 sets its state to IDLE. Then, at step 9(4)-8, for each job stored in current list 150, instruction processor 122 sends a signal to signal processor 124 (effectively transferring the jobs from current list 150 in [order of from first to last] to signal processor 124). The transfer of each such signal from current list 150 as seen by signal processor 124 as an event E(SP)2 [see Fig. 6 and action A(SP)2]. At step 9(4)-9, instruction processor 122 sends an EXIT signal to signal processor

124, which EXIT signal is viewed as Event E(SP)1 and responded to with action A(SP)1 by signal processor 124.

If the checks of steps 9(4)-1 and 9(4)-2 are negative, at step 9(4)-1-3, instruction processor 122 determines if current list 150 is not empty. If current list 150 is not empty, at step 9(4)-10 instruction processor 122 fetches the signal associated with the first job in current list 150 and begins executing that first job. After step 9(4)-10, action A(IP)4 terminates as indicated by symbol 9(4)-5.

If none of the above-described steps 9(4)-1, 9(4)-2, or 9(4)-3 are affirmative, at step 9(4)-4 instruction processor 122 sets its state to IDLE and sends an EXIT signal to signal processor 124, thereby indicating that there are no more jobs awaiting execution in current list 150. The EXIT signal is seen by signal processor 124 as Event E(SP)1 and responded to with action A(SP)1 by signal processor 124. After step 9(4)-4, action A(IP)4 terminates as indicated by symbol 9(4)-5.

It should be understood from the foregoing that execution can, in appropriate instances, continue in a looping operation as buffered signals in current list 150 are executed until all jobs in current list 150 are executed.

(9) IP RECEIVING SIGNAL FROM SP TO PUT IN CURRENT LIST

As mentioned above, instruction processor 122 can receive a signal from signal processor 124 that is to be put into current list 150. Such occurs particularly in performance by signal processor 124 of actions A(SP)1 [see, e.g., steps 8(1)-8, 8(1)-18, and 8(1)-20] and A(SP)4 [see, e.g., step 8(4)-9], and is viewed as event E(IP)9 by instruction processor 122. In response to event E(IP)9, instruction processor 122 performs action A(IP)9 which, as shown in Fig. 9(9), is merely the posting of the signal as the last signal on current list 150 (see step 9(9)-1).

(10) IP RECEIVING KILL SIGNAL FROM SP

As indicated previously, a "KILL" signal can be generated by signal processor 124, as occurs (for example) during a time period expiration (see action A(SP)3, e.g., step 8(3)-10). Such a KILL signal is seen as event E(IP)3 by instruction processor 122, and is responded to by action A(IP)3 as shown in Fig. 9(3). Upon receipt of a KILL signal, at step 9(3)-1 instruction processor 122 sets its state to IDLE. Then, as a result of the time period expiration (e.g., time out), at step 9(3)-2 instruction processor 122 essentially discards the current thread. At step 9(3)-3, instruction processor 122 sends an EXIT signal to signal processor 124. Such EXIT signal is seen as signal processor 124 as event E(SP)1, and is responded to with action A(SP)1 [see Fig. 8(1)] in the manner previously discussed.

(11) IP FORCED EXIT OF BUFFERED SIGNAL

A forced exit instruction of a buffered signal occurs when the application programmer needs to ensure that the exit really exists. That is, a small delay must be inserted so that other threads have the possibility to execute. When a forced exit of a buffered signal does occur, instruction processor 122 encounters event E(IP)7 and responds with action A(IP)7, the steps of which are depicted in Fig. 9(7). At step 9(7)-1, instruction processor 122 sends to signal processor 124 a signal for each job remaining in current list 150, thereby effectively transferring the contents of current list 150 to signal processor 124. Thereafter, instruction processor 122 sets its state to IDLE (step 9(7)-2). Then, at step 9(7)-3 instruction processor 122 sends an EXIT signal to signal processor 124. Such EXIT signal is seen as signal processor 124 as event E(SP)1, and is responded to with action A(SP)1 [see Fig. 8(1)] in the manner previously discussed.

Alternatively to the steps shown in Fig. 9(7), actions comparable to a forced exit can instead be accomplished by sending the signal with a time delay.

Fig. 5 shows another embodiment of the present invention, and particularly features central processing system 220 which includes a plurality of instruction processors 222_1, 222_2, ... 222_n in addition to signal processor 224, bus handlers 230, and a shared memory 227 (having both a program store and a data store). Each instruction processor 222 of the embodiment of Fig. 5 can

have the configuration above discussed with reference to Fig. 4.

Blocks executing in a multi-IP environment such as Fig. 5 must be able to handle several concurrent executions within a block, or otherwise it must be verifiable that only one processor is executing in a block at a time (e.g., semaphores or some means of synchronizing different threads is necessary). In a system 220 such as shown in Fig. 5, it is also possible to allocate certain blocks to only one of the instruction processors 222. In such configuration, such allocated instruction processor 222 has some of its data in a special fast memory, thereby making it possible to execute certain jobs very efficiently. As a trade-off, however, other instruction processors 222 would have to change context).

Thus, it is seen from the foregoing that instruction processors of the present invention are involved in the scheduling of jobs which it executes, primarily by the instruction processors themselves entering jobs associated with buffered signals on current list 150 (as occurs, for example, in step 9(11)-7 of action A(IP)11 [see Fig. 9(11)]). If the buffered signal happens to be generated immediately before exiting of the currently executing job, is of any but the lowest priority level, and if no interrupt is set, the job associated with the buffered signal is executed upon exit from the job in which the buffered signal is generated [see step 9(11)-6 of action A(IP)11 in Fig. 9(11)].

Subsequently, when the instruction processor exits from a job executed as a result of generation of a buffered signal, instruction processor can perform any one of several alternative steps depending upon priority level of the buffered signal and depending upon whether any interrupts have been set. A first such alternative includes obtaining the first (next) job from the current list and executing the same [see step 9(4)-10 of action A(IP)4 in Fig. 9(4)]. A second such alternative step includes returning scheduling control to signal processor 124, as occurs when the buffered signal is of a lowest priority level [see step 9(4)-6 of action A(IP)4 in Fig. 9(4)]. A third such alternative step includes transferring the entire contents of current list 150 to signal processor 124, as occurs when an interrupt has occurred [see step 9(4)-8 of action A(IP)4 in Fig. 9(4)].

In connection with the first alternative, consecutive jobs on the current list can be consecutively executed as jobs on the current list are exited. That is, when a job fetched from current list 150 (as at step 9(4)-10) is completed, its exit instruction creates (another) event E(IP)4 which, depending upon priority levels and interrupt status, can lead to execution of the next signal on current list 150.

As understood particularly from step 9(11)-5 of Fig. 9(11), whether or not a job associated with a buffered signal is executed immediately after the job which generates the buffered signal depends on the whether the buffered signal is generated immediately before a predetermined type of instruction (e.g, an EXIT

instruction). Thus, an instruction causing generation of a buffered signal must be in a predefined order within a job to have the buffered signal executed immediately upon termination of the job. Otherwise, the job associate with the buffered signal is subject to remaining on current list 150 and being executed subsequently to jobs associated with other buffered signals, or even being transferred to signal processor 124.

Whenever execution of a job by the instruction processor is interrupted, the entire contents of current list 150 is transferred to signal processor 124. The signal processor 124 then puts the signals from the current list 150 into appropriate ones of its buffers 142 in accordance with priority levels of the transferred signals.

In general, when an instruction processor needs a new signal for execution from the signal processor (as occurs upon receiving an EXIT signal from the instruction processor), the signal is fetched from buffers 142A - 142B in accordance with age (oldest) and priority level (i.e., level A, B, C, or D) [see Action A(SP)1 of Fig. 8(1)]. Flags "Active_priority" and CurrentThread_ID are established in accordance with the priority level and thread ID of the fetched signal, and the fetched signal is sent to the instruction processor [(see step 8(1)-19 in Fig. 8(1)]. Moreover, the signal processor sends to current list 150 any jobs in buffers 142A - 142B having the same thread ID as the fetched job [(see step 8(1)-20 in Fig. 8(1)], so that jobs having the same thread ID can be executed in proximity.

Thus, signal processor 124 also has the capability of sending signals to current list 150. Such particularly occurs when, as described above, the signal processor transfers to the current list another signal in its at least one buffer that has a same thread identification as the signal being transferred to the instruction processor. A similar posting of a signal on current list 150 occurs when signal processor 124 determines that the thread ID of the externally-generated signal (received via either a regional processor bus handler 30 or a processor bus handler 31) is the CurrentThreadID being processed by instruction processor 122 [see step A(SP)4-5 in Fig. 8(4)].

If the priority level of the fetched job as described above is either of the "C" priority level or "D" priority level, the signal processor determines whether any jobs of similar priority level are currently interrupted and, if so, resumes execution of the interrupted job while sending the fetched job back to the appropriate one of the buffers 142C or 142D. See Action A(SP)1 in Fig. 8(1).

Should the instruction processor be executing a level D signal at the time the signal processor receives a signal of higher priority (e.g, from the instruction processor, from a regional processor, or from an "other" instruction processor), execution of the level D signal by the instruction processor is immediately interrupted. [See, e.g., step 8(2)-7 in Fig. 8(2) and step 8(4)-8 in Fig. 8(4)].

In another embodiment, instruction processor 122 is provided with a plurality of current lists 150A - 150D, each one of the plurality of current lists being associated with a corresponding priority level. For example, jobs having an "A" priority level would be placed, where appropriate, on current list 150A; jobs having a "B" priority level would be placed, where appropriate, on current list 150B; and so forth.

It should be understood that variations of the foregoing may be implemented in view of particularly desired communications schemes. For example, with reference to event E(SP)1 [receiption of an EXIT signal from instruction processor 122] and action A(SP)1, a suitable alternate procedure might be implemented. In one such example, while anticipating an EXIT signal from instruction processor 122, signal processor 124 can go ahead and prepare a signal for instruction processor 122 beforehand. Then, when instruction processor 122 comes to the EXIT point (at which it is supposed to request information from signal processor 124), instruction processor 122 can check whether there are any prepared messages waiting for it. In such procedure, instruction processor 122 need not wait for signal processor 124 at an EXIT situation.

Advantageously, by provision of features such as current list 150, central processing systems according to the present invention can effectively employ very fast memory (e.g., cache memory), since central processing systems of the present invention do not change context at a rate to nullify effects of such fast memories. In

particular, the instruction processors of the present invention are able to reuse "cached" data. This means that data previously accessed in a thread can be reused. Also, by fetching data from primary memory (e.g., DRAM), a buffer of more than one data word can be fetched at a time. Thus, much of the main memory access can be implemented as cache accesses, thereby reducing the load on the instruction processor.

Moreover, central processing systems according to the present invention relieve signal processor 124 from some of its work, as instruction processor 122 itself schedules, in current list 150, jobs associated with buffered signals. In some normal traffic mix implementations (with 1 ms time-out), a significant work load reduction is realized for signal processor 124.

Further, since in the present invention buffered signals are often executed immediately when they are generated, thus decreasing the load of communication with signal processor 124, whereby instruction processor 122 also has a decreased work load.

In systems configured according to the present invention, the clock frequency of instruction processor 122 need not be bounded by the clock frequency of signal processor 124.

Central processing systems of the present invention handles buffered signal scheduling faster than the described prior art method. Even though the number of main memory accesses decreases with the present

invention, the memory bandwidth increases since instruction processor 122 will fetch much larger blocks of data at each main memory access.

Since, according to the present invention, there is more time between fetching of jobs from job buffers 142A- 142D by the signal processor, the signal processor has more time to prepare those jobs. Accordingly, the instruction processor stays in the IDLE state for a shorter time, when leaving control over to the signal processor, to fetch the next thread.

The present invention is particularly beneficial for execution-intensive tasks that do not frequently interact with regional processors, e.g., transit switches based on CCITT No. 7 signaling, Home Location Registers (HLRs), and Service Control Points (SCPs). To support such applications, the present invention advantageously provides externally-generated signals with thread IDS [see step A(SP)4-1 in Fig. 6A], so that jobs belonging to the same thread are executed as one thread, thereby increasing the number of jobs which execute between context switches.

The present invention also makes feasible emulation of certain devices, such as a central processor of an Ericsson AXE 10 switch, on RISC workstations, particularly since those workstations are provided with cache memories.

Moreover, it should be understood that the principles of the present invention do not necessarily

require separate processors for the signal processor and the instruction processor. In this regard, the present invention encompasses a single processor which emulates the distinct functions as herein described of both a signal processor and an instruction processor.

While the invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that various alterations in form and detail may be made therein without departing from the spirit and scope of the invention.

TABLE OF ELEMENTS 1410-13

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A processing system in which a signal processor schedules jobs to be executed by an instruction processor and transmits a job-associated signal to the instruction processor when the instruction processor requires a next job for execution, wherein the improvement comprises:

a current list memory maintained by the instruction processor, and wherein when a current job executed by the instruction processor causes the instruction processor to generate a buffered signal associated with a new job to be executed, the instruction processor causes the buffered signal associated with the new job to be stored in the current list.

2. The system of claim 1, wherein the instruction processor selectively stores the buffered signal associated with the new job in the current list in accordance with a priority level of the current job which caused the buffered signal to be generated.

3. The system of claim 1, wherein the instruction processor selectively stores the buffered signal associated with the new job in the current list in accordance with whether the signal processor has issued an interrupt to the instruction processor.

4. The system of claim 1, wherein the new job is immediately executed if an instruction in the current job

which generated the buffered signal is in a predefined order within the current job.

5. The system of claim 4, wherein the new job is immediately executed if the instruction in the current job which generated the buffered signal immediately precedes an EXIT instruction of the current job.

6. The system of claim 1, wherein the instruction processor causes the buffered signal associated with the new job to be stored in a predefined location in the current list.

7. The system of claim 6, wherein the instruction processor causes the buffered signal associated with the new job to be stored as the last job in the current list.

8. The system of claim 1, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor sends an EXIT signal to the signal processor if the job associated with the buffered signal has a predefined priority level.

9. The system of claim 8, wherein the predefined priority level is a lowest priority level.

10. The system of claim 1, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor sends all remaining jobs in the current list to the signal processor if the signal processor has issued an interrupt to the instruction processor.

11. The system of claim 1, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor fetches and executes a further job from the current list.

12. The system of claim 1, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor selective does one of the following:

(A) sends an EXIT signal to the signal processor if the job associated with the buffered signal has a predefined priority level;

(B) sends all remaining jobs in the current list to the signal processor if the signal processor has issued an interrupt to the instruction processor; or

(C) fetches and executes a further job from the current list.

13. The system of claim 1, wherein the signal processor is emulated by the instruction processor which performs the jobs scheduled by the signal processor.

14. The system of claim 1, wherein the signal processor selectively stores signals on the current list.

15. The system of claim 14, wherein when the signal processor comprises at least one buffer from which signals are fetched and transmitted to the instruction processor, and wherein when the signal processor transmits a signal to the instruction processor, the signal processor transfers to the current list another signal in its at least one buffer that has a same thread

identification as the signal being transferred to the instruction processor.

16. The system of claim 1, wherein a plurality of current list memories are maintained by the instruction processor.

17. The system of claim 16, wherein each of the plurality of current list memories is associated with a corresponding priority level.

18. A processing system in which a signal processor schedules jobs to be executed by an instruction processor and transmits a job-associated signal to the instruction processor when the instruction processor is to execute a job, the signal processor being operable to receive signals generated externally from without the processing system and from the instruction processor, wherein the improvement comprises:

the signal processor, when receiving a signal from the instruction processor or externally from without the processing system, setting an interrupt to the instruction processor when a priority level of received signal is of a highest priority level and exceeds a priority level of a current job being executed by the instruction processor.

19. The system of claim 18, wherein a current list memory is maintained by the instruction processor, wherein when a current job executed by the instruction processor causes the instruction processor to generate a buffered signal associated with a new job to be executed,

the instruction processor causes the buffered signal associated with the new job to be stored in the current list, and wherein when the priority level of an externally-generated signal is of the highest priority level and exceeds the priority level of the current job being executed by the instruction processor, contents of the current list are transferred to the signal processor.

20. The system of claim 18, wherein the signal processor interrupts the current job being executed by the instruction processor when the current job executed by the instruction processor is of a lowest level priority and the externally-generated signal has a priority level higher than the lowest level priority.

21. The system of claim 18, wherein the signal processor is emulated by the instruction processor which performs the jobs scheduled by the signal processor.

22. A method of operating a processing system in which a signal processor schedules jobs to be executed by an instruction processor and transmits a job-associated signal to the instruction processor when the instruction processor is to execute a job scheduled by the signal processor, wherein the improvement comprises:

the instruction processor maintaining a current list, so that when a current job being executed by the instruction processor causes the instruction processor to generate a buffered signal associated with a new job to be executed, the instruction processor causes the buffered signal associated with the new job to be stored in the current list.

23. The method of claim 22, wherein the instruction processor selectively stores the buffered signal associated with the new job in the current list in accordance with a priority level of the current job which caused the buffered signal to be generated.

24. The method of claim 22, wherein the instruction processor selectively stores the buffered signal associated with the new job in the current list in accordance with whether the signal processor has issued an interrupt to the instruction processor.

25. The method of claim 22, wherein the new job is immediately executed if an instruction in the current job which generated the buffered signal is in a predefined order within the current job.

26. The method of claim 23, wherein the new job is immediately executed if the instruction in the current job which generated the buffered signal immediately precedes an EXIT instruction of the current job.

27. The method of claim 22, wherein the instruction processor causes the buffered signal associated with the new job to be stored in a predefined location in the current list.

28. The method of claim 27, wherein the instruction processor causes the buffered signal associated with the new job to be stored as the last job in the current list.

29. The method of claim 22, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor sends an EXIT signal to the signal processor if the job associated with the buffered signal has a predefined priority level.

30. The method of claim 29, wherein the predefined priority level is a lowest priority level.

31. The method of claim 22, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor sends all remaining jobs in the current list to the signal processor if the signal processor has issued an interrupt to the instruction processor.

32. The method of claim 22, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor fetches and executes a further job from the current list.

33. The method of claim 22, wherein, when the instruction processor finishes execution of a job associated with a buffered signal, the instruction processor selective does one of the following:

(A) sends an EXIT signal to the signal processor if the job associated with the buffered signal has a predefined priority level;

(B) sends all remaining jobs in the current list to the signal processor if the signal processor has issued an interrupt to the instruction processor; or

(C) fetches and executes a further job from the current list.

34. The method of claim 22, wherein the signal processor is emulated by the instruction processor which performs the jobs scheduled by the signal processor.

35. The method of claim 22, wherein the signal processor selectively stores signals on the current list.

36. The method of claim 33, wherein when the signal processor comprises at least one buffer from which signals are fetched and transmitted to the instruction processor, and wherein when the signal processor transmits a signal to the instruction processor, the signal processor transfers to the current list another signal in its at least one buffer that has a same thread identification as the signal being transferred to the instruction processor.

37. The method of claim 22, wherein a plurality of current list memories are maintained by the instruction processor.

38. The method of claim 22, wherein each of the plurality of current list memories is associated with a corresponding priority level.

39. A method of operating an processing system in which a signal processor schedules jobs to be executed by an instruction processor and transmits a job-associated signal to the instruction processor when the instruction processor is to execute a job, the processing system being operable to receive signals generated externally from without the processing system and from the instruction processor, wherein the improvement comprises:

the signal processor, when receiving a signal from the instruction processor or externally from without the processing system, setting an interrupt to the instruction processor when a priority level of an externally-generated signal is of a highest priority level and exceeds a priority level of a current job being executed by the instruction processor.

40. The method of claim 39, wherein the instruction processor maintains a current list memory, wherein, when a current job executed by the instruction processor causes the instruction processor to generate a buffered signal associated with a new job to be executed, the instruction processor causes the buffered signal associated with the new job to be stored in the current list, and wherein, when the priority level of an externally-generated signal is of the highest priority level and exceeds the priority level of the current job being executed by the instruction processor, contents of the current list are transferred to the signal processor.

41. The method of claim 39, wherein the signal processor interrupts the current job being executed by the instruction processor when the current job executed by

the instruction processor is of a lowest level priority and the externally-generated signal has a priority level higher than the lowest level priority.

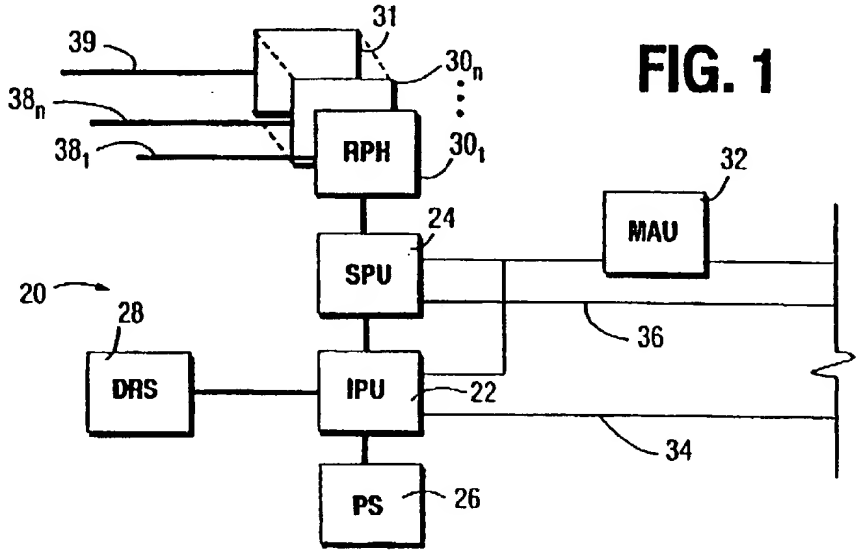
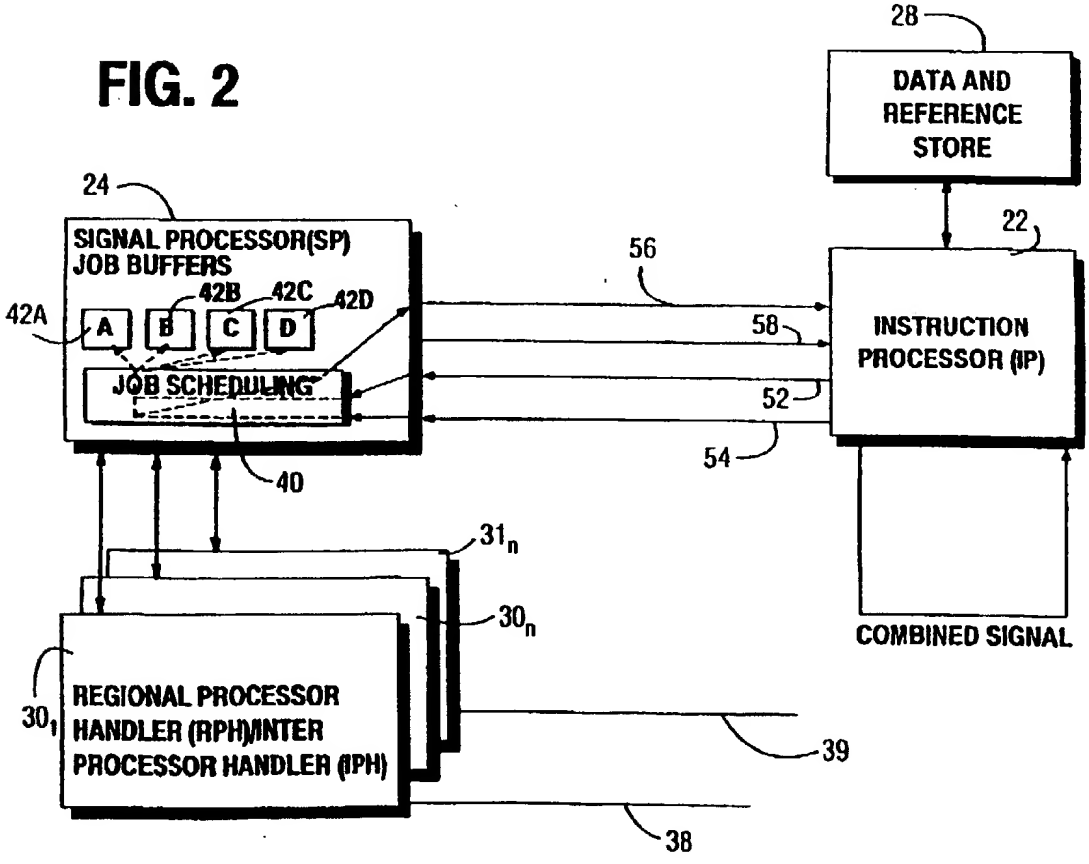

31
39
30n
38n
RPH
38₁
30₁
32
MAU
24
SPU
20
28
36
DRS
IPU
22
34
PS
26

FIG. 1

FIG. 2


28
DATA AND REFERENCE STORE
24
SIGNAL PROCESSOR(SP) JOB BUFFERS
22
56
42B
42C
42D
42A
A
B
C
D
58
INSTRUCTION PROCESSOR (IP)
JOB SCHEDULING
52
54
40
31n
30n
COMBINED SIGNAL
30₁
REGIONAL PROCESSOR HANDLER (RPH)/INTER PROCESSOR HANDLER (IPH)
39
38

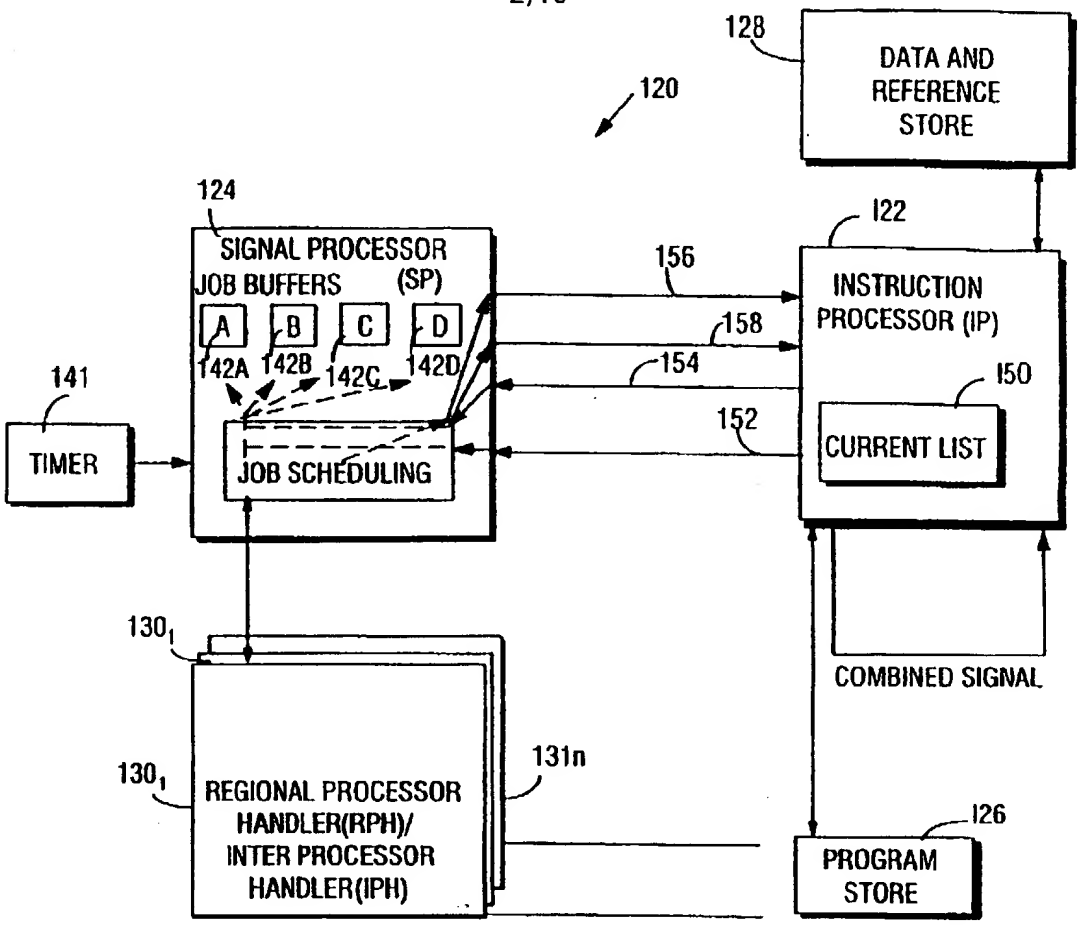


FIG. 3

THREAD ID	JOB LEVEL	FORMAT	SIGNAL NUMBER	REC BLOCK	SEND BLOCK	FORLOPP	REQ PRO	DATA LIST

FIG. 10

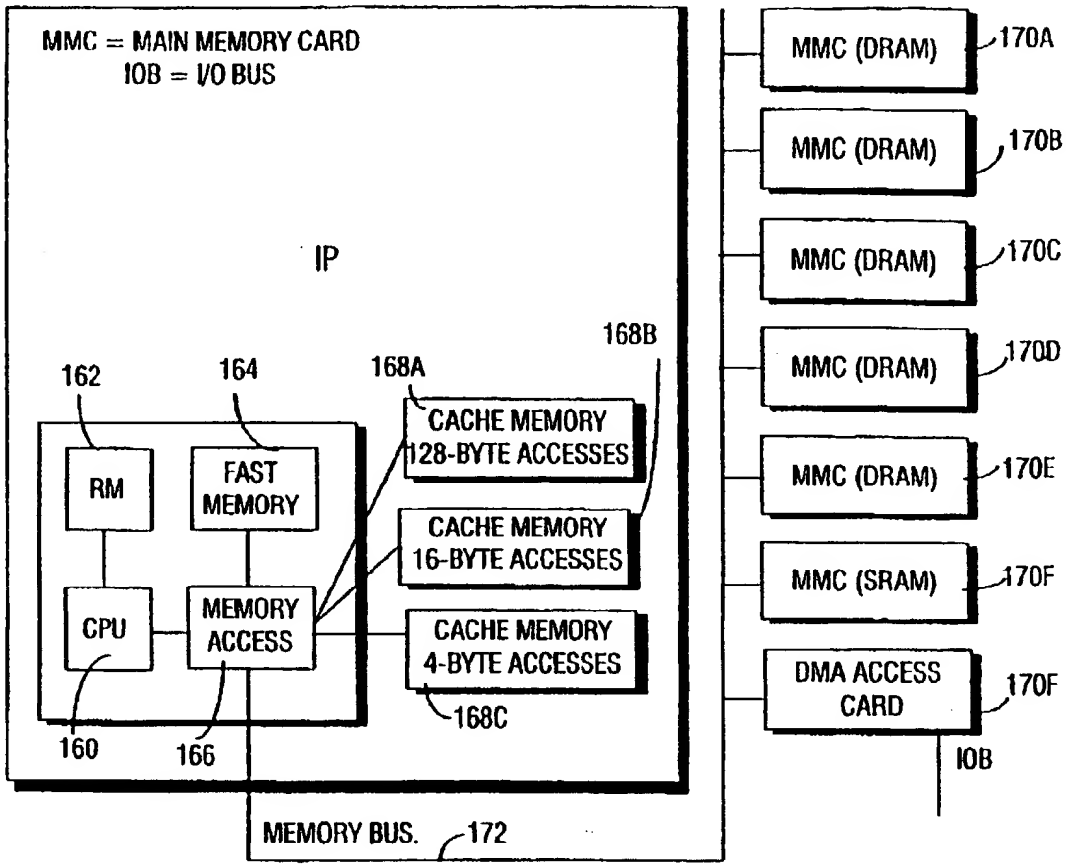

MMC = MAIN MEMORY CARD
IOB = I/O BUS
IP
162
164
168A
168B
RM
FAST MEMORY
CPU
MEMORY ACCESS
CACHE MEMORY 128-BYTE ACCESSES
CACHE MEMORY 16-BYTE ACCESSES
CACHE MEMORY 4-BYTE ACCESSES
168C
160
166
MEMORY BUS.
172
MMC (DRAM)
170A
MMC (DRAM)
170B
MMC (DRAM)
170C
MMC (DRAM)
170D
MMC (DRAM)
170E
MMC (SRAM)
170F
DMA ACCESS CARD
170F
IOB

FIG.4

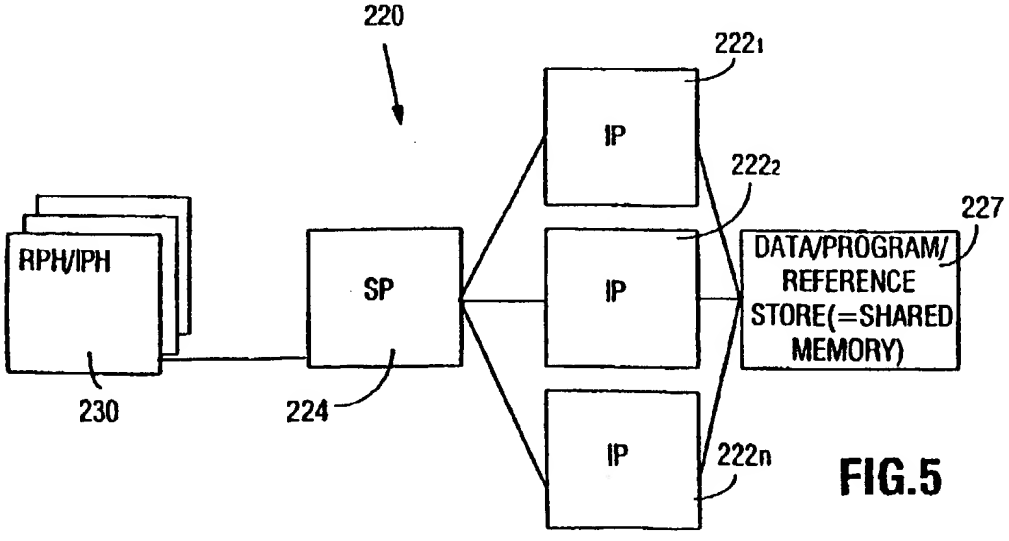

220
222₁
IP
222₂
227
RPH/IPH
SP
IP
DATA/PROGRAM/ REFERENCE STORE(=SHARED MEMORY)
230
224
IP
222n

FIG.5

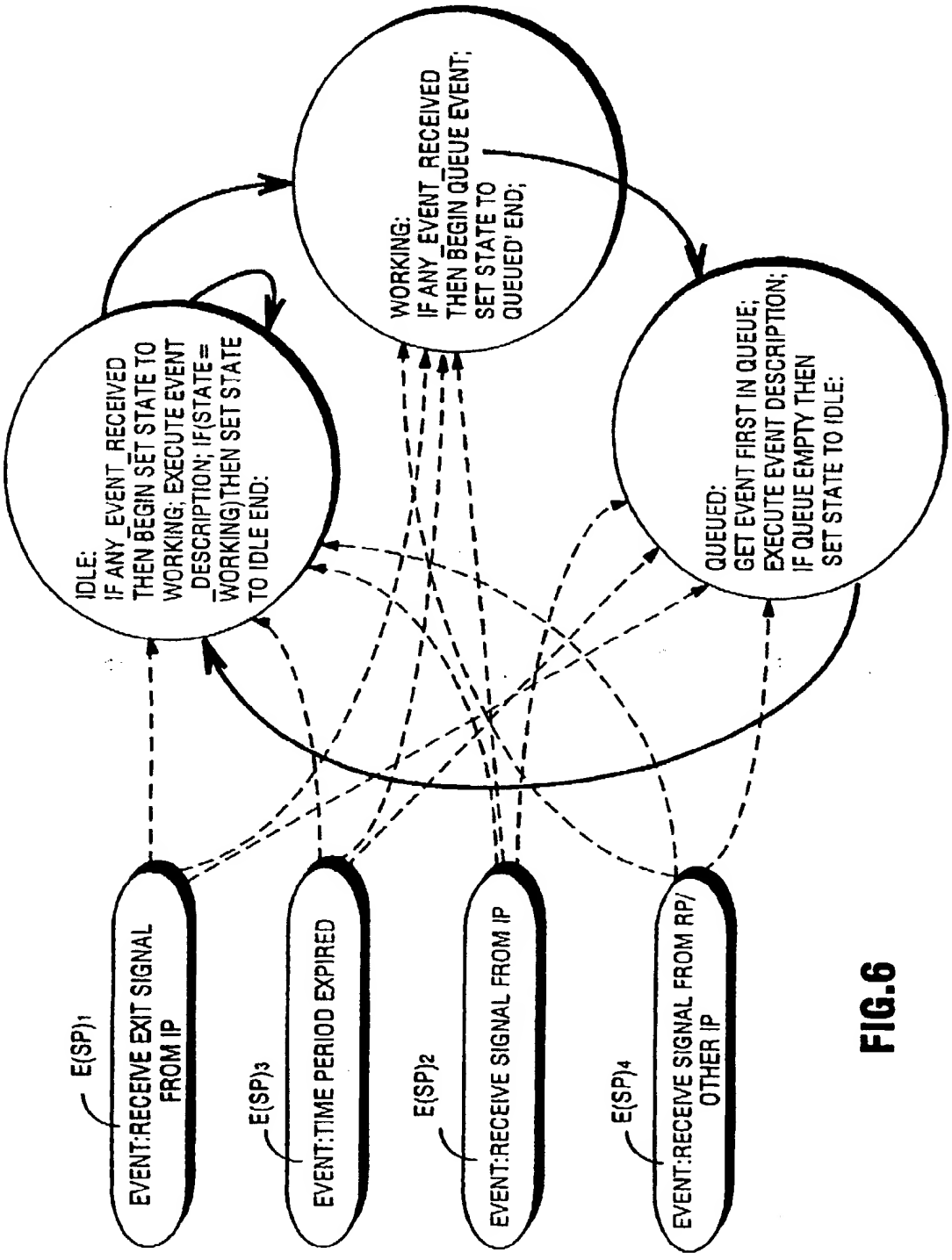

IDLE:
IF ANY_EVENT_RECEIVED
THEN BEGIN SET STATE TO
WORKING; EXECUTE EVENT
_DESCRIPTION; IF(STATE=
_WORKING)THEN SET STATE
TO IDLE END;
WORKING:
IF ANY_EVENT_RECEIVED
THEN BEGIN QUEUE EVENT;
SET STATE TO
QUEUED' END;
QUEUED:
GET EVENT FIRST IN QUEUE;
EXECUTE EVENT DESCRIPTION;
IF QUEUE EMPTY THEN
SET STATE TO IDLE;
E(SP)1
EVENT:RECEIVE EXIT SIGNAL
FROM IP
E(SP)3
EVENT:TIME PERIOD EXPIRED
E(SP)2
EVENT:RECEIVE SIGNAL FROM IP
E(SP)4
EVENT:RECEIVE SIGNAL FROM RP/
OTHER IP

FIG.6

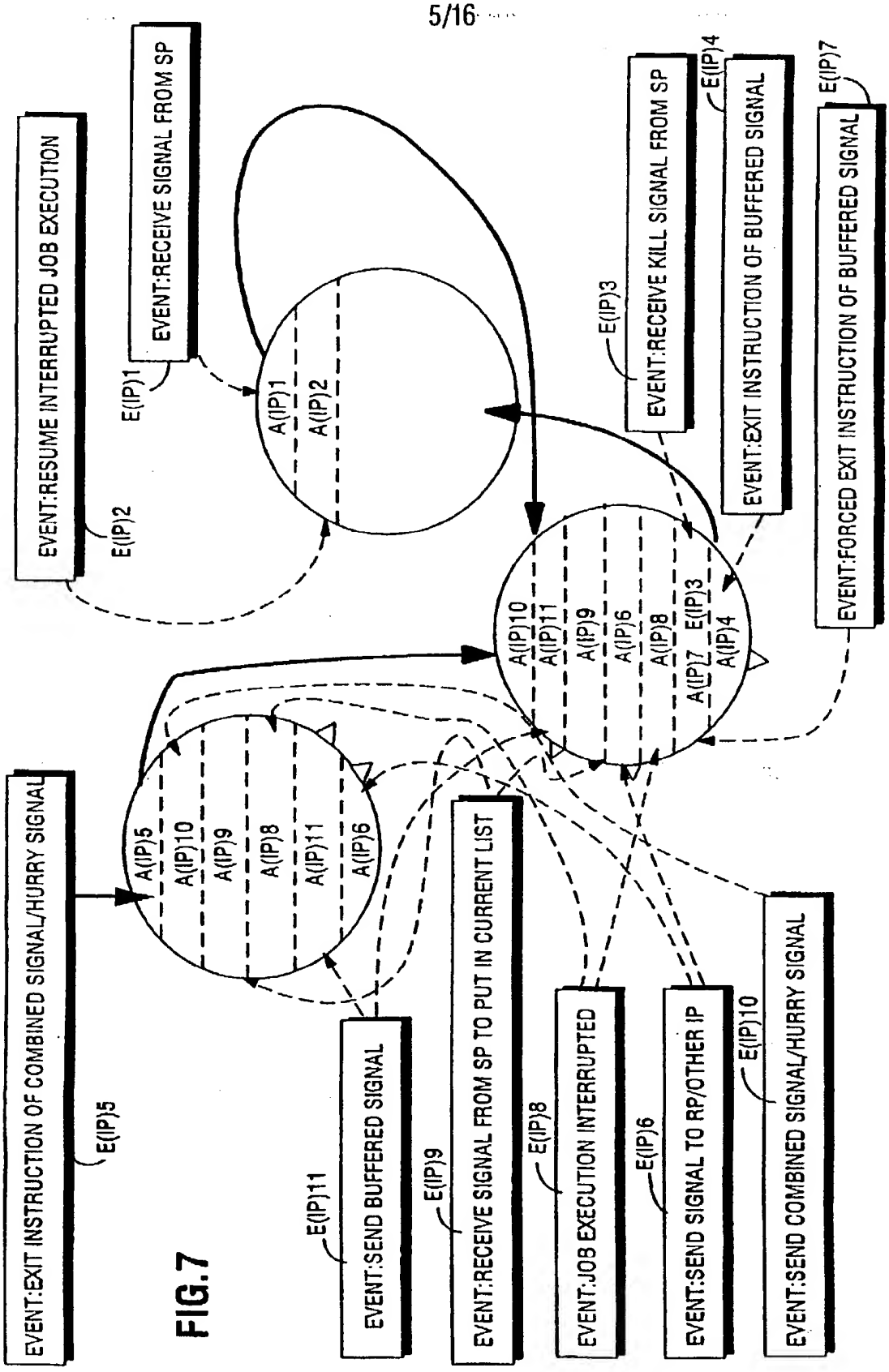

EVENT:EXIT INSTRUCTION OF COMBINED SIGNAL/HURRY SIGNAL
E(IP)5
EVENT:RESUME INTERRUPTED JOB EXECUTION
E(IP)2
EVENT:RECEIVE SIGNAL FROM SP
E(IP)1
EVENT:RECEIVE KILL SIGNAL FROM SP
E(IP)3
EVENT:EXIT INSTRUCTION OF BUFFERED SIGNAL
E(IP)4
EVENT:FORCED EXIT INSTRUCTION OF BUFFERED SIGNAL
E(IP)7
A(IP)1
A(IP)2
A(IP)5
A(IP)10
A(IP)9
A(IP)8
A(IP)11
A(IP)6
A(IP)10
A(IP)11
A(IP)9
A(IP)6
A(IP)8
A(IP)7
E(IP)3
A(IP)4
EVENT:SEND BUFFERED SIGNAL
E(IP)11
EVENT:RECEIVE SIGNAL FROM SP TO PUT IN CURRENT LIST
E(IP)9
EVENT:JOB EXECUTION INTERRUPTED
E(IP)8
EVENT:SEND SIGNAL TO RP/OTHER IP
E(IP)6
EVENT:SEND COMBINED SIGNAL/HURRY SIGNAL
E(IP)10

FIG.7

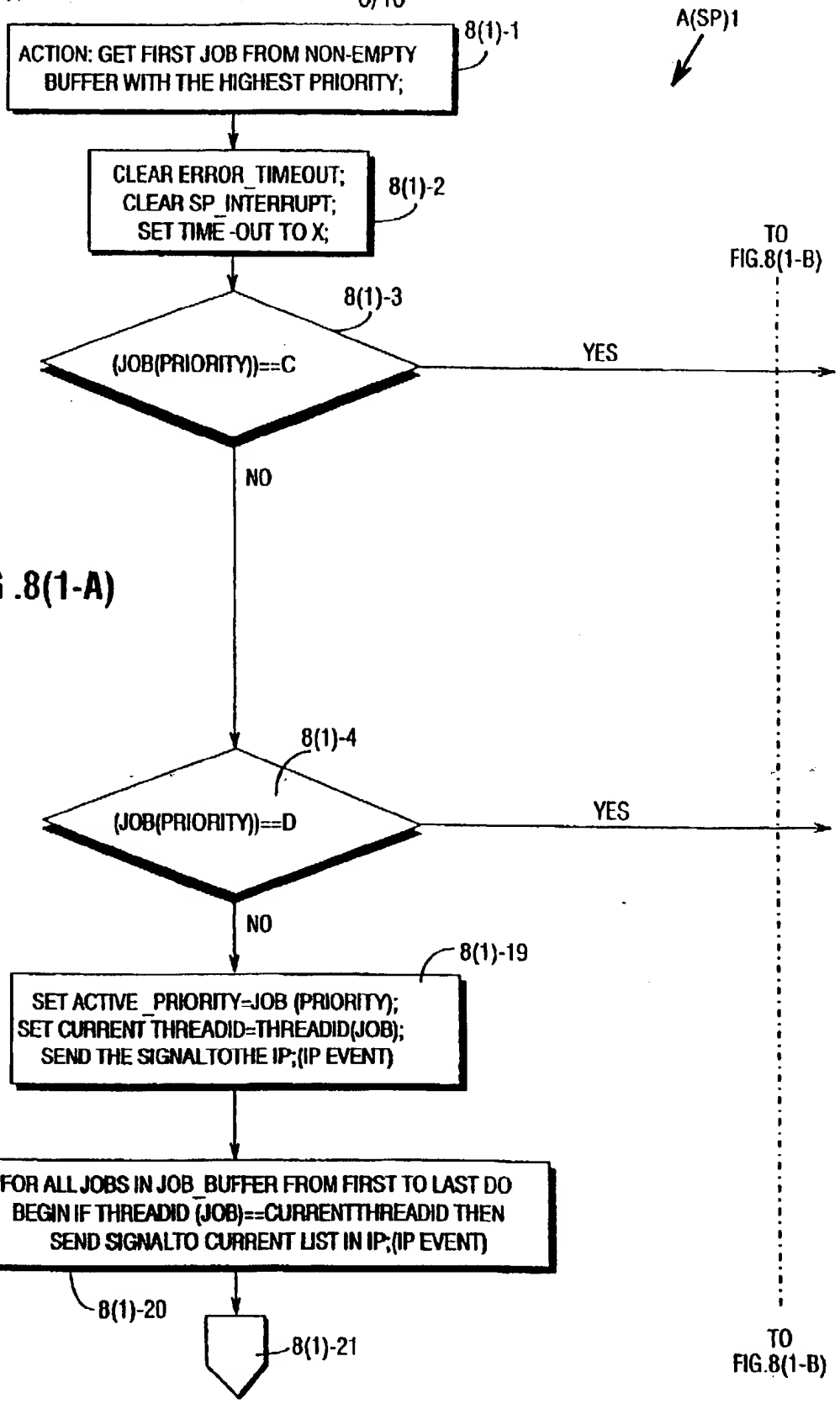

A(SP)1
ACTION: GET FIRST JOB FROM NON-EMPTY BUFFER WITH THE HIGHEST PRIORITY;
8(1)-1
CLEAR ERROR_TIMEOUT;
CLEAR SP_INTERRUPT;
SET TIME -OUT TO X;
8(1)-2
TO FIG.8(1-B)
8(1)-3
(JOB(PRIORITY))==C
YES
NO
8(1)-4
(JOB(PRIORITY))==D
YES
NO
8(1)-19
SET ACTIVE_PRIORITY=JOB (PRIORITY);
SET CURRENT THREADID=THREADID(JOB);
SEND THE SIGNALTOTHE IP;(IP EVENT)
FOR ALL JOBS IN JOB_BUFFER FROM FIRST TO LAST DO
BEGIN IF THREADID (JOB)==CURRENTTHREADID THEN
SEND SIGNALTO CURRENT LIST IN IP;(IP EVENT)
8(1)-20
8(1)-21
TO FIG.8(1-B)

FIG .8(1-A)

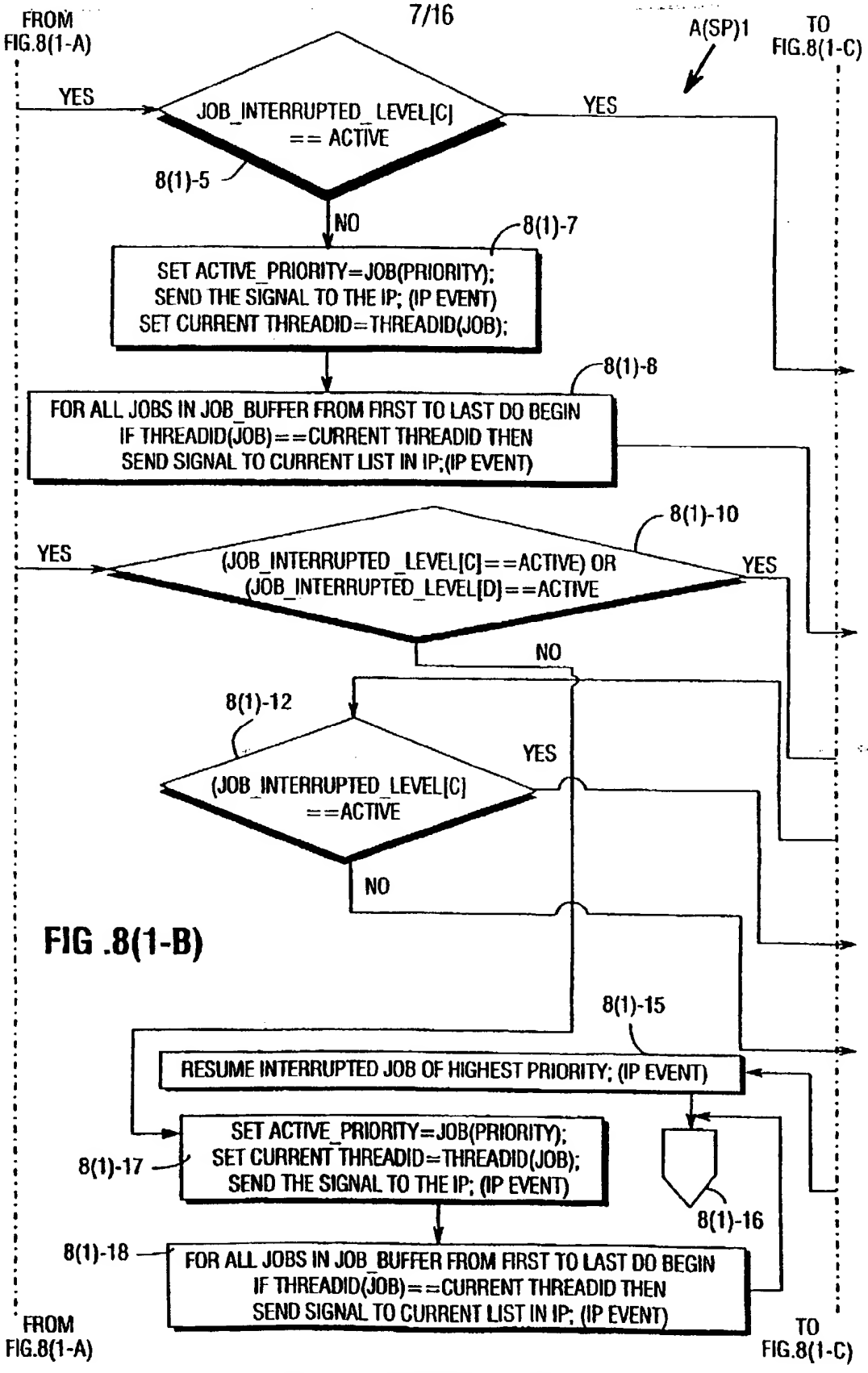
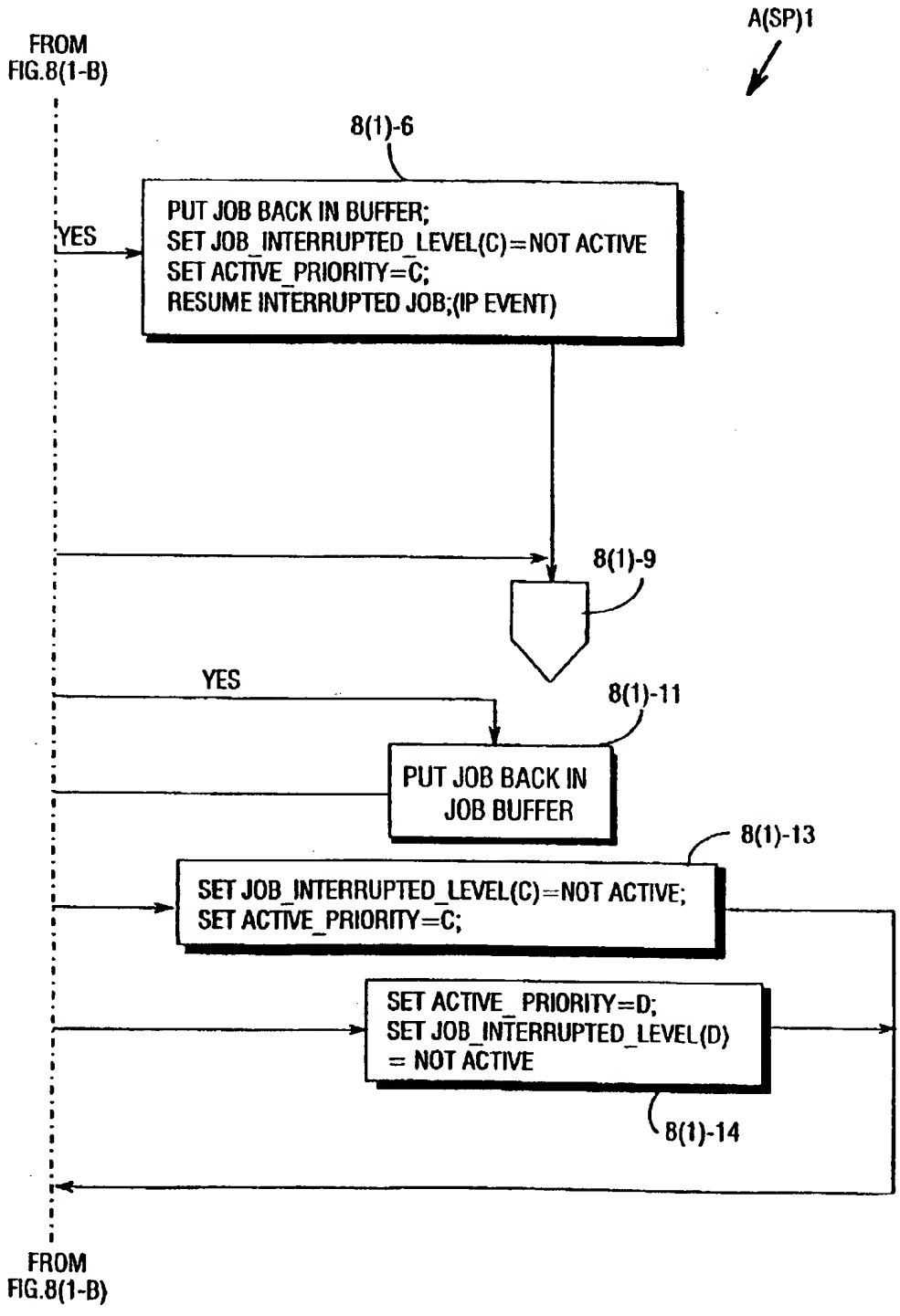

FROM FIG.8(1-A)
A(SP)1
TO FIG.8(1-C)
YES
JOB_INTERRUPTED_LEVEL[C] == ACTIVE
YES
8(1)-5
NO
8(1)-7
SET ACTIVE_PRIORITY=JOB(PRIORITY);
SEND THE SIGNAL TO THE IP; (IP EVENT)
SET CURRENT THREADID=THREADID(JOB);
8(1)-8
FOR ALL JOBS IN JOB_BUFFER FROM FIRST TO LAST DO BEGIN
IF THREADID(JOB)==CURRENT THREADID THEN
SEND SIGNAL TO CURRENT LIST IN IP;(IP EVENT)
8(1)-10
YES
(JOB_INTERRUPTED_LEVEL[C]==ACTIVE) OR
(JOB_INTERRUPTED_LEVEL[D]==ACTIVE
YES
NO
8(1)-12
YES
(JOB_INTERRUPTED_LEVEL[C]
==ACTIVE
NO
8(1)-15
RESUME INTERRUPTED JOB OF HIGHEST PRIORITY; (IP EVENT)
SET ACTIVE_PRIORITY=JOB(PRIORITY);
SET CURRENT THREADID=THREADID(JOB);
SEND THE SIGNAL TO THE IP; (IP EVENT)
8(1)-17
8(1)-16
8(1)-18
FOR ALL JOBS IN JOB_BUFFER FROM FIRST TO LAST DO BEGIN
IF THREADID(JOB)==CURRENT THREADID THEN
SEND SIGNAL TO CURRENT LIST IN IP; (IP EVENT)
FROM FIG.8(1-A)
TO FIG.8(1-C)

FIG .8(1-B)

FIG. 8(1-C)


A(SP)1
FROM FIG.8(1-B)
8(1)-6
PUT JOB BACK IN BUFFER;
SET JOB_INTERRUPTED_LEVEL(C)=NOT ACTIVE
SET ACTIVE_PRIORITY=C;
RESUME INTERRUPTED JOB;(IP EVENT)
YES
8(1)-9
YES
8(1)-11
PUT JOB BACK IN JOB BUFFER
8(1)-13
SET JOB_INTERRUPTED_LEVEL(C)=NOT ACTIVE;
SET ACTIVE_PRIORITY=C;
SET ACTIVE_ PRIORITY=D;
SET JOB_INTERRUPTED_LEVEL(D) = NOT ACTIVE
8(1)-14
FROM FIG.8(1-B)

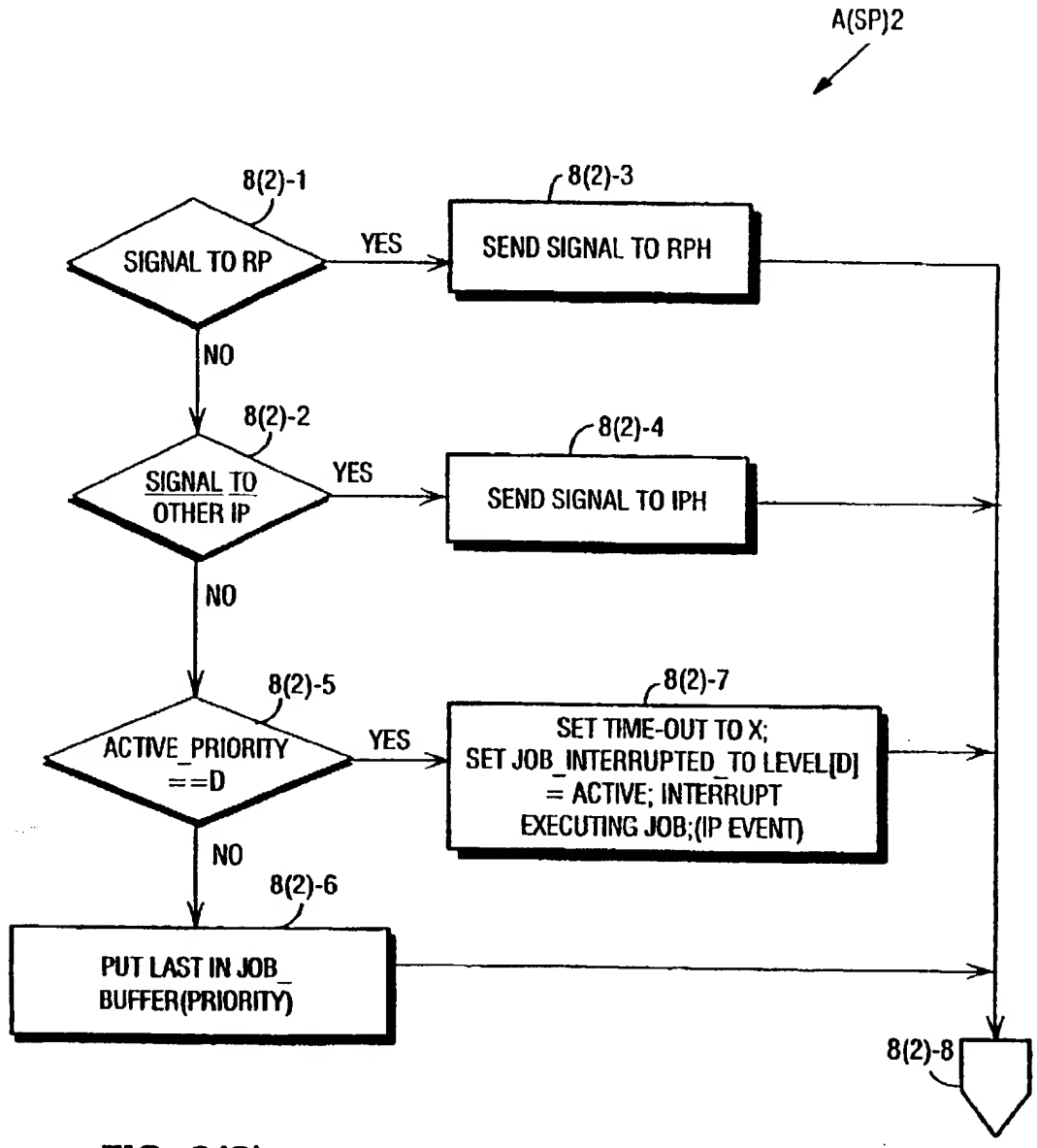

A(SP)2
8(2)-1
SIGNAL TO RP
YES
8(2)-3
SEND SIGNAL TO RPH
NO
8(2)-2
SIGNAL TO OTHER IP
YES
8(2)-4
SEND SIGNAL TO IPH
NO
8(2)-5
ACTIVE_PRIORITY ==D
YES
8(2)-7
SET TIME-OUT TO X; SET JOB_INTERRUPTED_TO LEVEL[D] = ACTIVE; INTERRUPT EXECUTING JOB;(IP EVENT)
NO
8(2)-6
PUT LAST IN JOB_ BUFFER(PRIORITY)
8(2)-8

FIG .8(2)

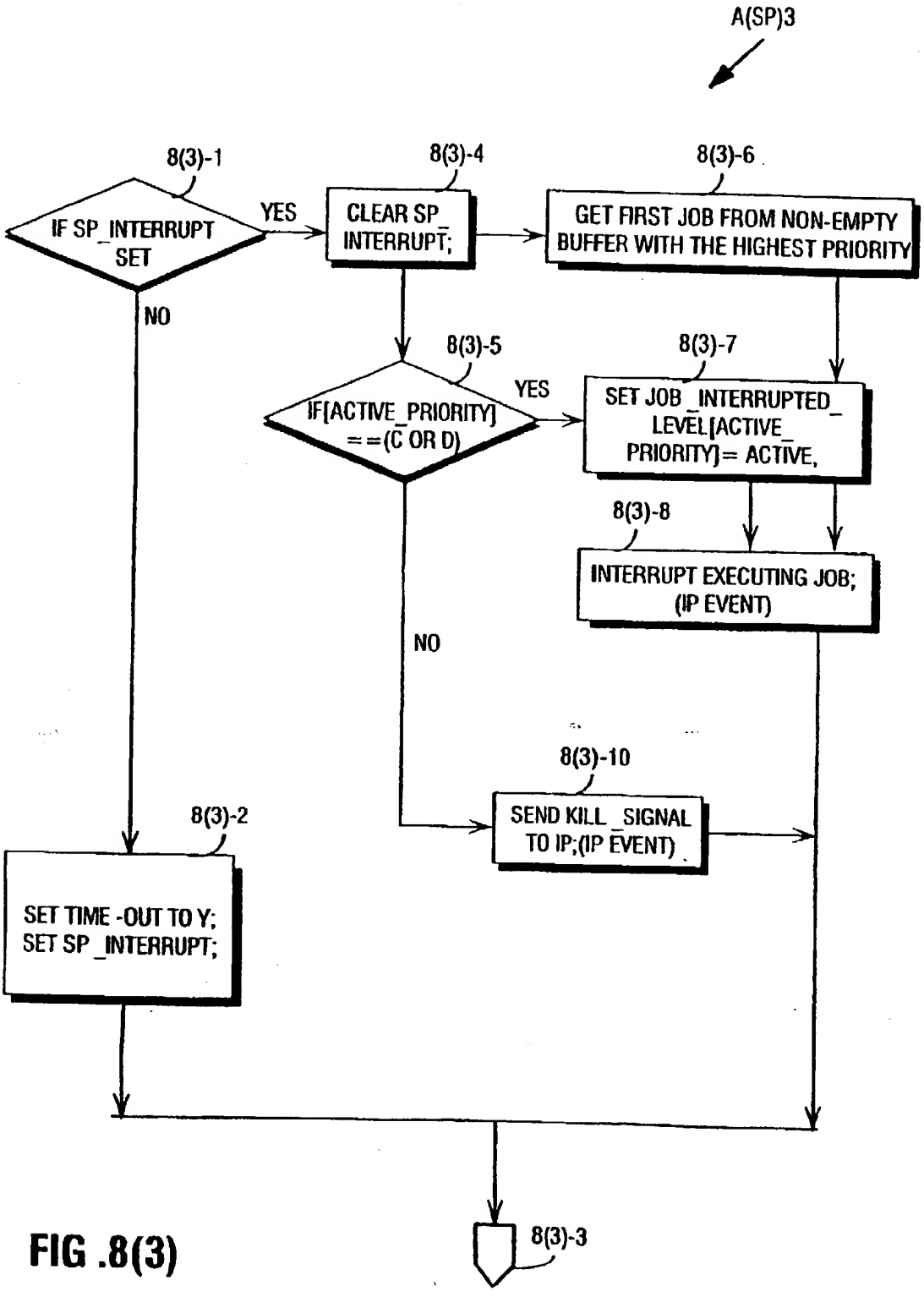

A(SP)3
8(3)-1
IF SP_INTERRUPT SET
YES
NO
8(3)-4
CLEAR SP_ INTERRUPT;
8(3)-6
GET FIRST JOB FROM NON-EMPTY BUFFER WITH THE HIGHEST PRIORITY
8(3)-5
IF[ACTIVE_PRIORITY] ==(C OR D)
YES
NO
8(3)-7
SET JOB_INTERRUPTED_ LEVEL[ACTIVE_ PRIORITY]= ACTIVE,
8(3)-8
INTERRUPT EXECUTING JOB; (IP EVENT)
8(3)-10
SEND KILL_SIGNAL TO IP;(IP EVENT)
8(3)-2
SET TIME -OUT TO Y; SET SP_INTERRUPT;
8(3)-3

FIG .8(3)

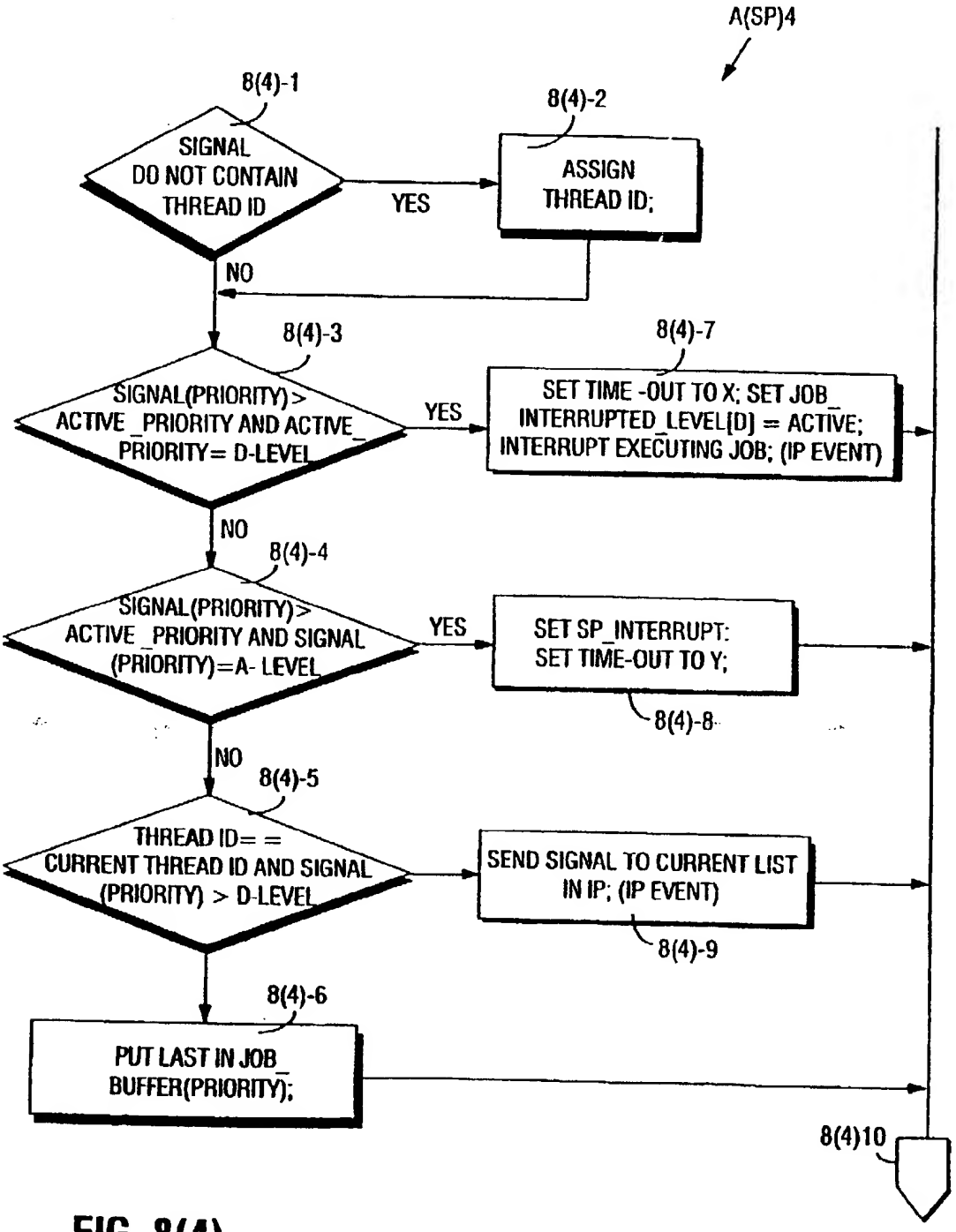

A(SP)4
8(4)-1
SIGNAL DO NOT CONTAIN THREAD ID
YES
8(4)-2
ASSIGN THREAD ID;
NO
8(4)-3
SIGNAL(PRIORITY)> ACTIVE _PRIORITY AND ACTIVE_ PRIORITY= D-LEVEL
YES
8(4)-7
SET TIME -OUT TO X; SET JOB_ INTERRUPTED_LEVEL[D] = ACTIVE; INTERRUPT EXECUTING JOB; (IP EVENT)
NO
8(4)-4
SIGNAL(PRIORITY)> ACTIVE _PRIORITY AND SIGNAL (PRIORITY)=A- LEVEL
YES
SET SP_INTERRUPT: SET TIME-OUT TO Y;
8(4)-8
NO
8(4)-5
THREAD ID= = CURRENT THREAD ID AND SIGNAL (PRIORITY) > D-LEVEL
SEND SIGNAL TO CURRENT LIST IN IP; (IP EVENT)
8(4)-9
8(4)-6
PUT LAST IN JOB_ BUFFER(PRIORITY);
8(4)10

FIG .8(4)

FIG .9(1)

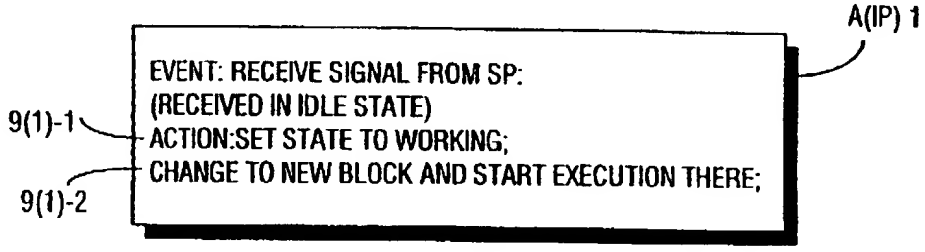


FIG .9(2)

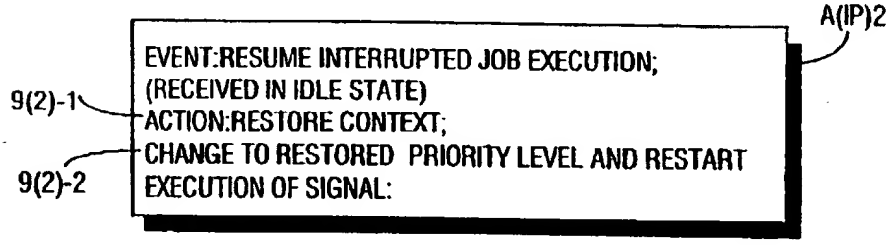
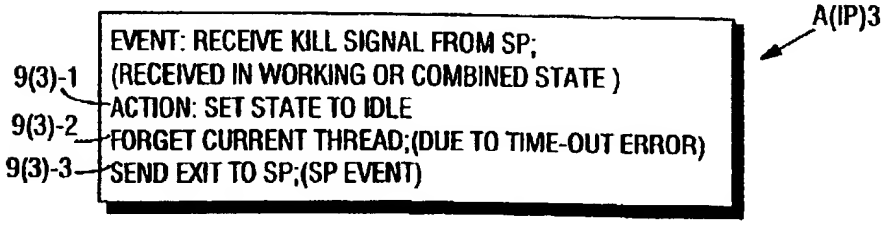


FIG .9(3)



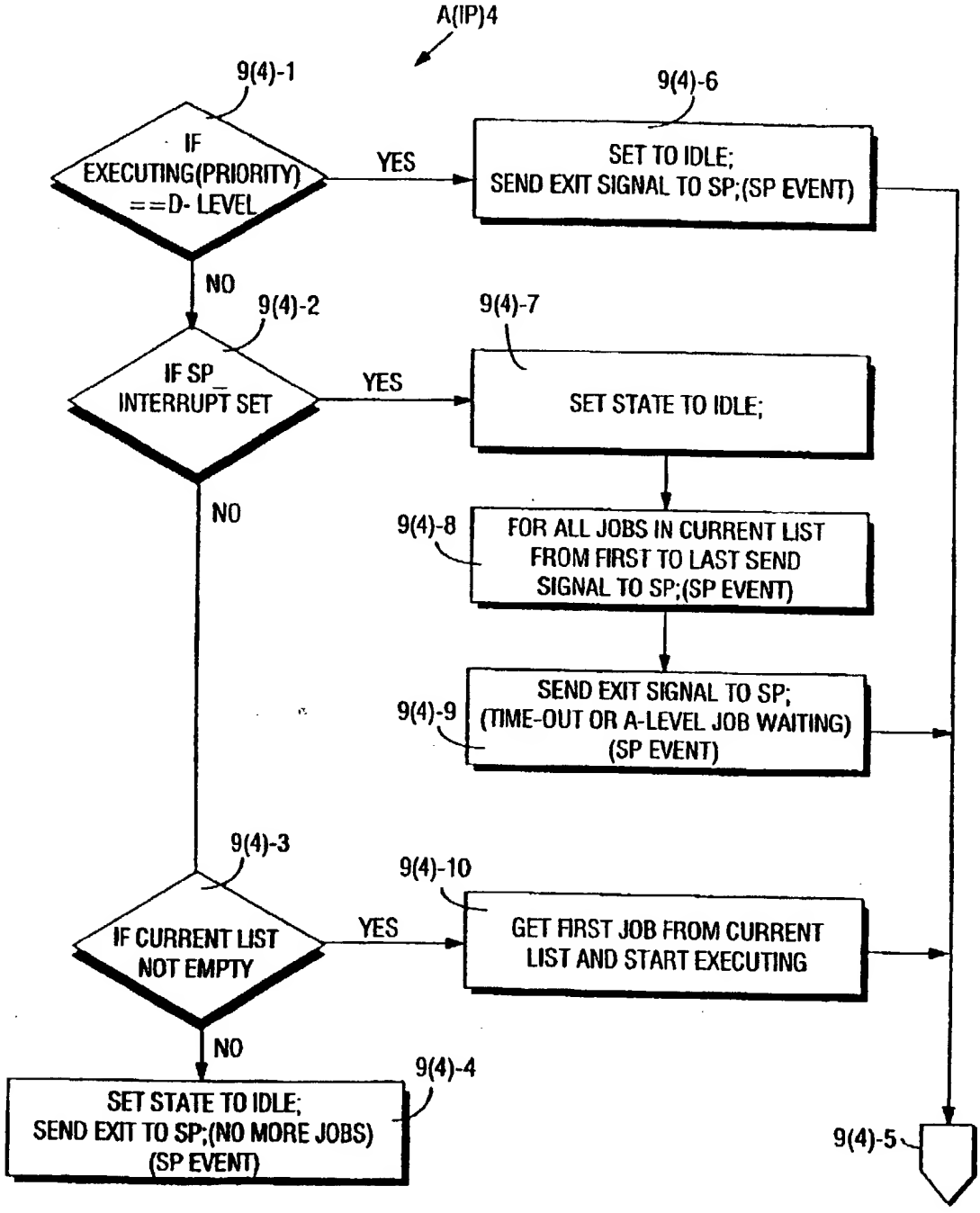

A(IP)4
9(4)-1
IF EXECUTING(PRIORITY) ==D- LEVEL
YES
9(4)-6
SET TO IDLE; SEND EXIT SIGNAL TO SP;(SP EVENT)
NO
9(4)-2
IF SP_ INTERRUPT SET
YES
9(4)-7
SET STATE TO IDLE;
NO
9(4)-8
FOR ALL JOBS IN CURRENT LIST FROM FIRST TO LAST SEND SIGNAL TO SP;(SP EVENT)
9(4)-9
SEND EXIT SIGNAL TO SP; (TIME-OUT OR A-LEVEL JOB WAITING) (SP EVENT)
9(4)-3
IF CURRENT LIST NOT EMPTY
YES
9(4)-10
GET FIRST JOB FROM CURRENT LIST AND START EXECUTING
NO
9(4)-4
SET STATE TO IDLE; SEND EXIT TO SP;(NO MORE JOBS) (SP EVENT)
9(4)-5

FIG. 9(4)

FIG .9(5)

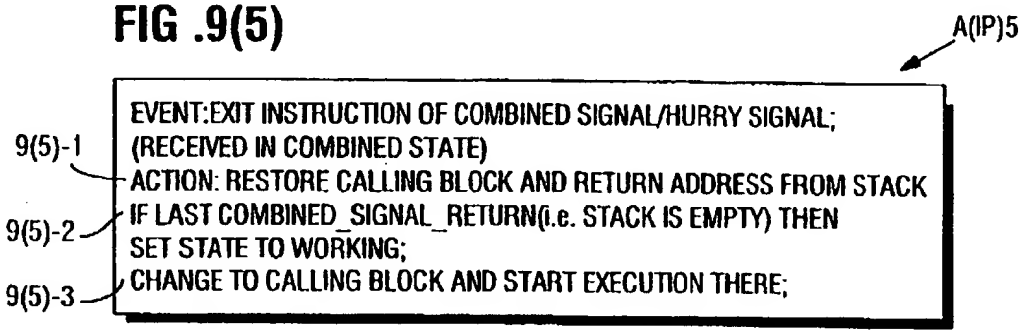


FIG .9(6)

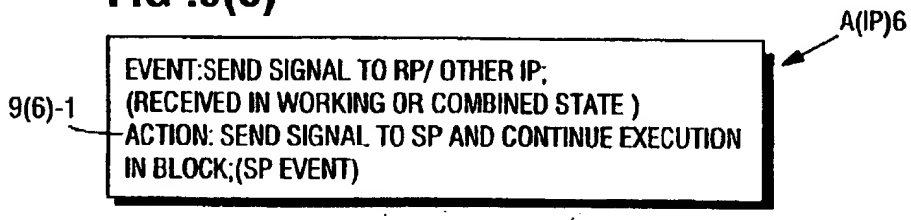


FIG .9(7)

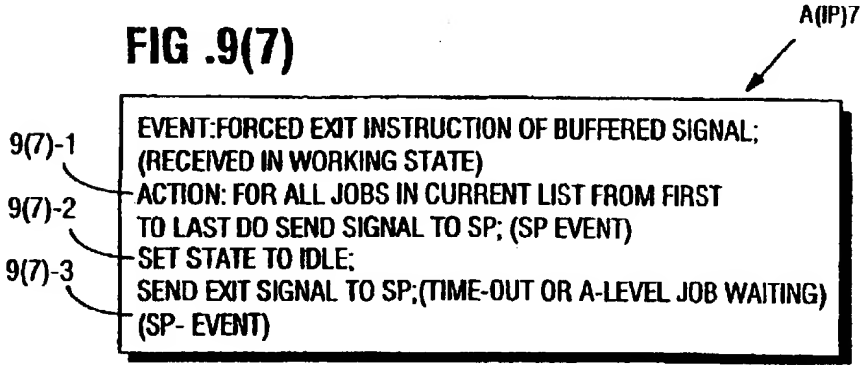


FIG .9(8)

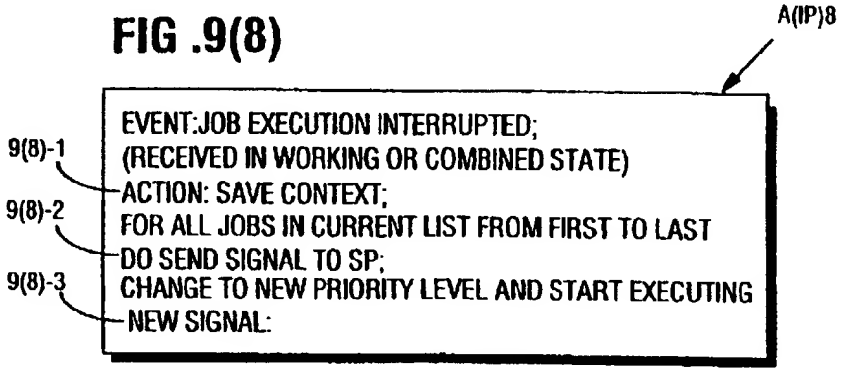


FIG .9(9)

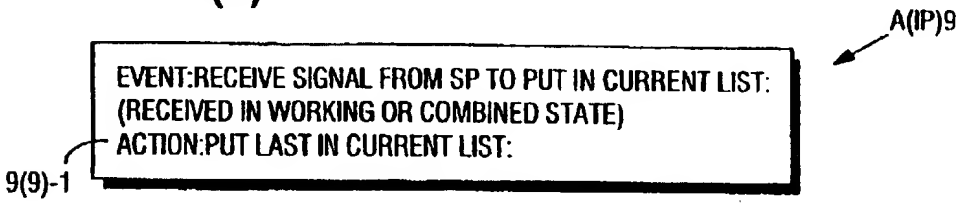
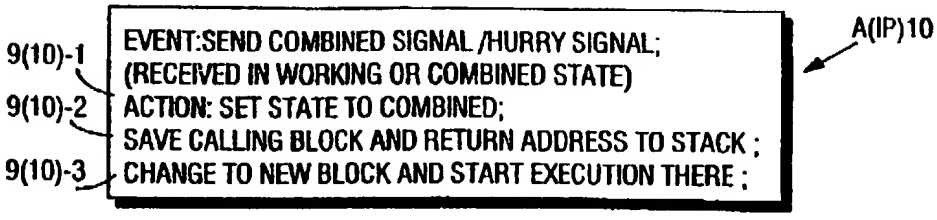


FIG .9(10)



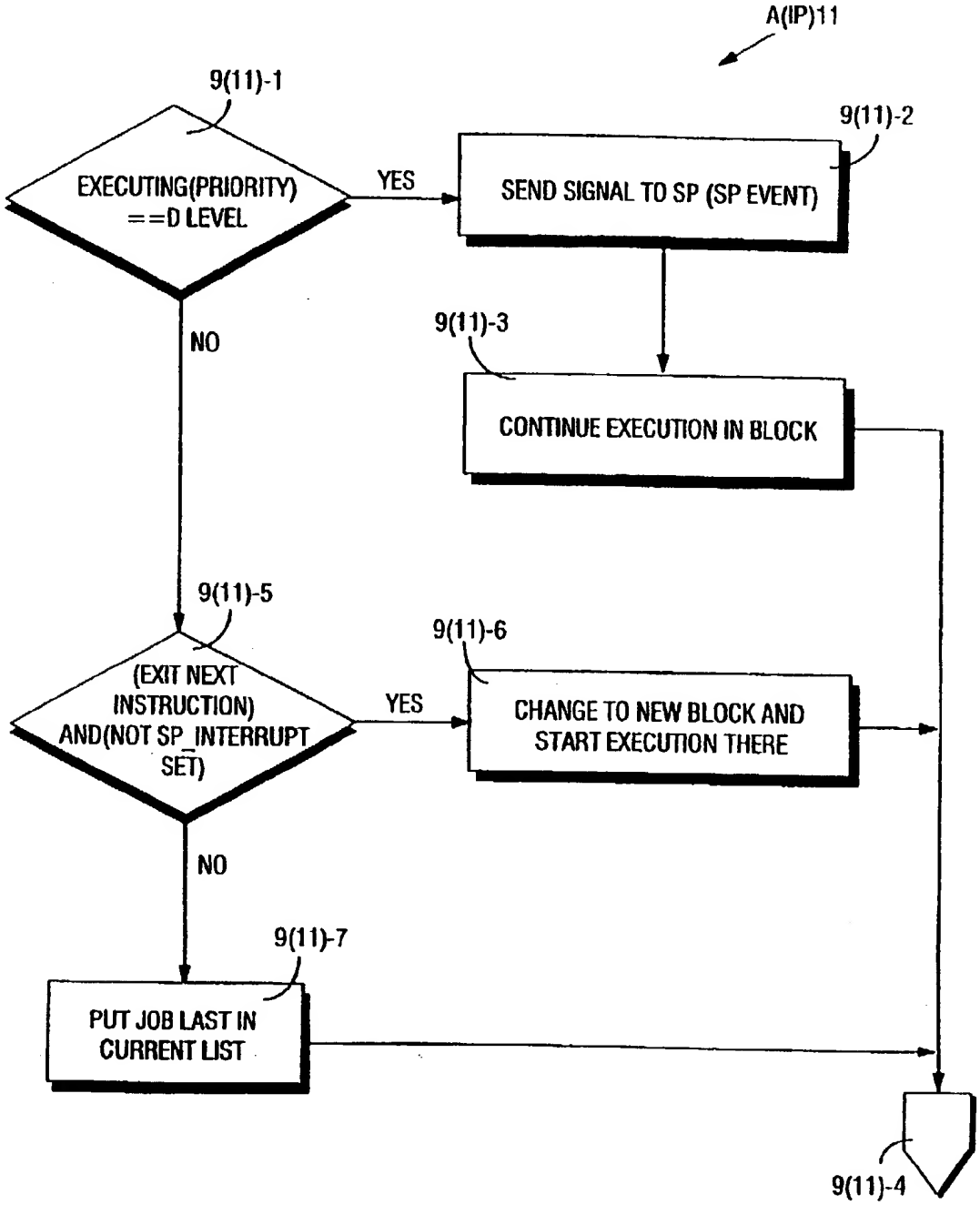

A(IP)11
9(11)-1
EXECUTING(PRIORITY) ==D LEVEL
YES
9(11)-2
SEND SIGNAL TO SP (SP EVENT)
NO
9(11)-3
CONTINUE EXECUTION IN BLOCK
9(11)-5
(EXIT NEXT INSTRUCTION) AND(NOT SP_INTERRUPT SET)
YES
9(11)-6
CHANGE TO NEW BLOCK AND START EXECUTION THERE
NO
9(11)-7
PUT JOB LAST IN CURRENT LIST
9(11)-4

FIG. 9(11)

INTERNATIONAL SEARCH REPORT

Intern. ial Application No
PCT/SE 96/01706

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	MICROPROCESSORS AND MICROSYSTEMS, vol. 18, no. 10, December 1994, ISSN 0141-9331, UK, pages 571-578, XP000488045 J.E. COOLING: "Task scheduling in hard real-time embedded systems using hardware co-processors" see abstract	18,39
A	see page 573, left-hand column, line 15 - page 574, left-hand column, line 26; figures 6-13	1,22,40
A	EP 0 362 983 A (UNISYS CORPORATION) 11 April 1990 see abstract see column 2, line 20 - column 3, line 12; figure 1	1,18,22, 39

-/--

[X] Further documents are listed in the continuation of box C. [X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
13 March 1997	14. 04. 97
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+ 31-70) 340-3016	Authorized officer Wiltink, J

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern al Application No
PCT/SE 96/01706

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 021 146 A (IBM) 7 January 1981 see abstract see page 1, line 1 - page 2, line 11; figures 4-1,4-2 see page 9, line 7 - page 10, line 4; figures 1-1,1-2	1,18,22, 39
A	WO 87 02486 A (BURROUGHS) 23 April 1987 see abstract see page 3, line 15 - page 4, line 21; figure 1	1,18,22, 39

1

INTERNATIONAL SEARCH REPORT

information on patent family members

Intern. al Application No
PCT/SE 96/01706

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 362903 A	11-04-90	US 4779194 A	18-10-88
		US 4796178 A	03-01-89
		EP 0364000 A	18-04-90
		CA 1289674 A	24-09-91
		CA 1306308 A	11-08-92
		CA 1299758 A	28-04-92
		DE 3650158 D	12-01-95
		DE 3650158 T	06-04-95
		DE 3650160 D	12-01-95
		DE 3650160 T	06-04-95
		EP 0243402 A	04-11-87
		JP 63501987 T	04-08-88
		WO 8702486 A	23-04-87
EP 21146 A	07-01-81	US 4286322 A	25-08-81
		BR 8004161 A	21-01-81
		JP 1385560 C	26-06-87
		JP 56011549 A	04-02-81
		JP 61055695 B	28-11-86
WO 8702486 A	23-04-87	US 4779194 A	18-10-88
		US 4796178 A	03-01-89
		CA 1289674 A	24-09-91
		CA 1306308 A	11-08-92
		CA 1299758 A	28-04-92
		DE 3650158 D	12-01-95
		DE 3650158 T	06-04-95
		DE 3650160 D	12-01-95
		DE 3650160 T	06-04-95
		EP 0243402 A	04-11-87
		EP 0362903 A	11-04-90
		EP 0364000 A	18-04-90
		JP 63501987 T	04-08-88

CLAIMS

1. In a computer system having a host computer coupled to a client computing device via a serial connection, an operating system embodied on a computer-readable medium at the host computer, comprising:

computer-executable instructions to listen at a first baud rate for a predefined message sent from the client computing device; and

computer-executable instructions to listen at a second baud rate for the predefined message in an event that the predefined message is not received at the first baud rate.

2. An operating system of claim 1, further comprising computer-executable instructions to listen at the first baud rate for a predetermined period.

3. An operating system of claim 1, further comprising computer-executable instructions to listen at the second baud rate for the predefined message in an event that error characters not forming part of the predefined message are received at the first baud rate.

4. An operating system of claim 1, further comprising computer-executable instructions to cache the second baud rate in an event that the predefined message is received at the second baud rate.

5. An operating system of claim 1, further comprising computer-executable instructions to look up the first and second baud rates in a table.

6. A computer comprising:

a processor; and

the operating system of claim 1, embodied on the computer-readable medium, and executed on the processor.

7. In a computer system having a host computer coupled to a client computing device via a serial connection, a computer program module embodied on a computer-readable medium for execution at the host computer, comprising:

computer-executable instructions to listen at a first baud rate at which a predefined message might be sent from the client computing device over the serial connection; and

computer-executable instructions to switch to listening at a second baud rate if one of the following events occurs: (1) characters not included in the predefined message are received, or (2) a predetermined timeout period expires without successful receipt of the predefined message.

8. A computer program module of claim 7, further comprising computer-executable instructions to cache one of the first and second baud rates at which the predefined message is successfully received.

9. An operating system incorporating the computer program module of claim 7.

10. A computer-implemented method, comprising:

listening at a first of multiple baud rates for a predefined message to be sent by a client computing device over a serial connection to a host computer;

in an event that characters not included as part of the message are received or the message is not detected within a predetermined time period, listening at a second of the baud rates for the message.

11. A computer-implemented method of claim 10, wherein the listening steps are repeated until a baud rate is found that allows receipt of the message.

12. A computer-implemented method of claim 11, further comprising storing the baud rate that enables receipt of the message.

13. A computer-implemented method of claim 10, further comprising storing the multiple baud rates in a table.

14. A computer-implemented method, comprising:

listening to a serial connection at a baud rate for a predefined message from a client computing device; and

automatically adjusting the baud rate in an event that the message is not detected.

15. A computer-implemented method of claim 14, wherein the adjusting comprises cycling through a set of predetermined baud rates.

16. A computer-implemented method of claim 14, further comprising caching the baud rate at which the message is detected.

17. In a computer system having a host computer coupled to a client computing device via a serial connection and employing a Unimodem null serial protocol to establish a connection between the host computer and the client computing device, a computer-implemented method, comprising:

(a) storing multiple baud rates at which a predefined message may be sent from the client computing device over the serial connection;

(b) selecting one of the baud rates;

(c) listening at the selected baud rate for the predefined message;

(d) in an event that the predefined message is not received, selecting another of the baud rates; and

(e) repeating steps (c) and (d) until a baud rate is found that enables receipt of the predefined message.